VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Využití metod strojového učení pro rozpoznávání řeči

Machine Learning Methods for Speech Recognition

2019 Bc. Martina Slívová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Martina Slívová**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: Využití metod strojového učení pro rozpoznávání řeči
Machine Learning Methods for Speech Recognition

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem diplomové práce je návrh a implementace systému pro automatické rozpoznávání řeči s využitím metod strojového učení.
Tento systém bude sloužit jako prvotní ochrana u komplexního autentizačního systému, kde bude jeho úkolem rozpoznání proneseného hesla.

Body zadání:
1. Detailně nastudujte problematiku automatického rozpoznávání řeči.
2. Proveďte rešerši stávajících systémů automatického rozpoznávání řeči.
3. Popište používané parametry u těchto systémů.
3. Popište metody strojového učení využívané při implementaci systémů automatického rozpoznávání řeči.
5. Navrhněte a implementujte systém pro rozpoznávání izolovaných slov ve vhodném vývojovém prostředí.
6. K navrženému systému vytvořte programovou a uživatelskou dokumentaci.

Seznam doporučené odborné literatury:

[1] PSUTKA, Josef. Mluvíme s počítačem česky. Praha: Academia, 2006. Česká matice technická (Academia). ISBN 80-200-1309-1.
[2] UHLÍŘ, Jan. Technologie hlasových komunikací. Praha: Nakladatelství ČVUT, 2007. ISBN 978-80-01-03888-8.
[3] LI, Jinyu, Li DENG, Reinhold HAEB-UMBACH a Yifan GONG. Robust automatic speech recognition: a bridge to practical applications. ISBN 0128023988.
[4] DENG, Li. a Douglas O'SHAUGHNESSY. Speech processing: a dynamic and optimization-oriented approach. New York: Marcel Dekker, c2003. ISBN 978-0824740405.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavol Partila, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

prof. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 18. 4. 2019 .....

Ráda bych na tomto místě poděkovala vedoucímu diplomové práce Ing. Pavolu Partilovi, Ph.D. za jeho odborné rady, konzultace a vstřícný přístup při vytváření této práce.

Abstrakt

Tato diplomová práce se zabývá využitím metod strojového učení pro rozpoznávání řeči. První část se věnuje teoretickému popisu metod zpracování řečového signálu a algoritmům, které je možné použít pro automatické rozpoznávání řeči. Jsou zde popsány algoritmy dynamického borcení času, skryté Markovovy modely a hluboké neuronové sítě. Praktická část byla věnována realizaci ASR systému pro izolovaná slova, který je založen na konvolučních neuronových sítích. Byly použity nástroje Keras a TensorFlow pro programovací jazyk Python. Pro trénování a testování byla použita databáze 15 slov, obsahující promluvy od žen a můžu v různém prostředí. Využití systému je možné různých oblastech Industry 4.0.

Klíčová slova: automatické rozpoznávání řeči; strojové učení; skryté Markovovy modely; dynamické borcení času; neuronové sítě, hluboké neuronové sítě; konvoluční neuronové sítě; Python; Keras; TensorFlow; Industry 4.0

Abstract

This thesis is devoted to machine learning methods for speech recognition. The first part deals with teoretical description of methods for speech signal processing and algorithms which can be used for automatic speech recognition. Dynamic time warping, hidden Markov models and deep neural networks are described here. The practical part is focused on the description of the created system, which is based on convolutional neural networks. This system was designed and implemented in Python using Keras and TensorFlow. A dataset of 15 words was used for training and testing. The use of the system is possible in various areas of Industry 4.0.

Key Words: automatic speech recognition; machine learning; hidden Markov models; dynamic time warping; neural networks; deep neural networks; convolutional neural networks; Python; Keras; TensorFlow; Industry 4.0

Obsah

Seznam použitých zkratek a symbolů **9**

Seznam obrázků **11**

Seznam tabulek **12**

Seznam výpisů zdrojového kódu **13**

Úvod **14**

1 Rešerše systémů využívajících automatické rozpoznávání řeči **15**

2 Zpracování řeči **17**
2.1 Tvorba řeči 17

3 Zpracování řečového signálu **19**
3.1 Předzpracování řečového signálu 19
3.2 Charakteristiky řeči 21

4 Automatické rozpoznávání řeči **26**
4.1 Metoda dynamického borcení času 27
4.2 Skryté Markovovy modely 28
4.3 Hluboké neuronové sítě 30

5 Použité nástroje **33**
5.1 Keras 33
5.2 TensorFlow 33
5.3 Datová sada 34

6 Implementace **36**
6.1 Načítání dat 36
6.2 Předzpracování 40
6.3 Ukládání dat 44
6.4 Model CNN 45
6.5 Trénování 48
6.6 Rozpoznávání 52

Závěr **55**

Literatura **56**

Přílohy **59**

A Příloha v IS EDISON **60**

B Programová a uživatelská dokumentace **61**

Seznam použitých zkratek a symbolů

ASR	–	Automatic Speech Recognition – automatické rozpoznávání řeči
MFCC	–	Mel-frequency Cepstral Coefficients – Mel-frekvenční kepstrální koeficienty
DTW	–	Dynamic Time Warping – dynamické borcení času
DNN	–	Deep Neural Network – hluboké neuronové sítě
DBN	–	síť typu Deep Belief Network
KNN	–	K-Nearest Neighbors algorithm – algoritmus k-nejbližších sousedů
VQ	–	Vector Quantization – vektorová kvantizace
HMM	–	Hidden Markov Model – skryté Markovovy modely
VAD	–	Voice Activity Detector – detektor hlasové aktivity
CNN	–	Covolutional Neural Netword – konvoluční neuronová síť
KWS	–	KeyWord Spotting – vyhledávání klíčových slov
ReLu	–	Rectified Linear Units – aktivační funkce
ANN	–	Artificial Neural Network – Umělé neuronové sítě
$s(t)$	–	signál spojitý v čase
$s[n]$	–	diskrétní signál
$S(f)]$	–	spektrum signálu $s(t)$
$g(t)$	–	budící funkce
$G(f)$	–	spektrum budící funkce
$h(t)$	–	impulzní odezva
$H(f)$	–	spektrum impulzní odezvy
T_{vz}	–	vzorkovací perioda [s]
f_{vz}	–	vzorkovací frekvence [Hz]
f_{max}	–	maximální frekvence [Hz]
t	–	čas [s]
SS složka	–	Stejnosměrná složka
μ_s	–	Stejnosměrná složka
FIR	–	Finite Impulse Response – filtr s konečnou impulzní odezvou
$w[n]$	–	váhovací funkce
L	–	počet vzorků
N	–	délka signálu
N_{ram}	–	počet rámců
s_{ram}	–	posun rámce
l_{ram}	–	délka rámce
p_{ram}	–	překrytí rámce
LPC	–	Linear Predictive Coding – lineární prediktivní kódování

E_n	–	krátkodobá energie
M_n	–	krátkodobá intenzita
Z_n	–	počet průchodů nulou
sgn	–	znaménková funkce
$H(z)$	–	přenosová funkce
$A(z)$	–	polynom proměnné z^{-1}
c_k	–	kepstrální koeficienty
f_m	–	frekvence [mel]
f	–	frekvence [Hz]
B_{mw}	–	Šířka pásma v melovské stupnici [mel]
B_w	–	Šířka pásma [Hz]
FFT	–	Fast Fourier Transform – rychlá Fourierova transformace
y_m	–	odezva filtru
$M*$	–	počet pásem melovského filtru
M	–	počet melovských kepstrálních koeficientů

Seznam obrázků

2.1 Hlasový trakt [13] . . . 18
2.2 Číslicový model hlasového traktu . . . 18
3.1 Předzpracování řeči . . . 19
3.2 Předzpracování řeči . . . 20
3.3 Výpočet MFCC . . . 23
3.4 Banka trojúhelníkových filtrů: a) v melovské škále, b) v původní frekvenční škále[13]. 24
4.1 Princip rozpoznávání izolovaných slov . . . 26
4.2 DTW – první fáze (trénování) . . . 27
4.3 DTW - druhá fáze (rozpoznávání) . . . 28
4.4 Příklad struktury HMM [18] . . . 29
4.5 Blokové schéma pro rozpoznávání izolovaných slov pomocí HMM . . . 29
4.6 Princip konvoluční vrstvy [23] . . . 31
4.7 Příklad funkce pooling vrstvy [23] . . . 31
4.8 Příklad architektury CNN [26] . . . 32
5.1 Lokální a distribuovaná struktura . . . 34
6.1 Architektura použité CNN . . . 47
6.2 Výpis nápovědy pro použití trénovacího programu . . . 49
6.3 Průběh trénování při 50 epochách . . . 50
6.4 Confusion matrix pro trénovací data . . . 51
6.5 Confusion matrix pro testovací data . . . 52
6.6 Confusion matrix pro validační data . . . 52
6.7 Vzhled grafické aplikace pro rozpoznávání . . . 54

Seznam tabulek

3.1 Typické hodnoty počtu filtrů M^* pro dané přenášené pásmo [13]. 24
5.1 Celkový přehled slov použité datové sady 35

Seznam výpisů zdrojového kódu

6.1 Načtení názvů slov . . . 36
6.2 Načtení zvukových souborů . . . 37
6.3 Načtení názvů slov ze souboru . . . 37
6.4 Načtení předzpracovaných dat ze souboru . . . 38
6.5 Načítaní dat . . . 39
6.6 Funkce pro odstranění stejnosměrné složky . . . 40
6.7 Preemfáze . . . 40
6.8 Segmentace . . . 41
6.9 Váhovací funkce . . . 41
6.10 FFT . . . 41
6.11 MFCC . . . 42
6.12 Výpočet dynamických koeficientů . . . 43
6.13 Spuštění předzpracování . . . 43
6.14 Načtení, předzpracování a uložení příznaků do souboru . . . 44
6.15 Uložená seznamu slov do souboru . . . 44
6.16 Model neuronové sítě . . . 45
6.17 Parser vstupních parametrů . . . 48
6.18 Rozdělení dat a spuštění trénování . . . 49
6.19 Vypsání confusion matrix . . . 51
6.20 Rozpoznávání z mikrofonu . . . 53

Úvod

Viditelným trendem dnešní doby je snaha o zjednodušení komunikace mezi člověkem a strojem. K tomuto účelu slouží systémy automatické rozpoznávání řeči, které extrahují obsah z řečového signálu. Tato úloha však není snadná, protože řeč je velmi komplexní. Takovéto systémy je možné použít například v průmyslu pro zadávání příkazů strojům nebo pro ovládání palubního počítače automobilu. Dalšími příklady může být i pomoc hendikepovaným, ovládání mobilního telefonu, převod řeči na text nebo složitější dialogové systémy.

Cílem této diplomové práce je návrh a implementace systému pro automatické rozpoznávání izolovaných slov. V první kapitole jsou popsány stávající systémy pro ASR (Automatic Speech Recognition) a jsou zde uvedeny jaké metody pro ně byly použity. Na základě tohoto popisu bude vybráno jakým způsobem bude vytvořen vlastní program pro ASR. V další kapitole je obecně popsána produkce lidské řeči a její matematický model.

Třetí kapitola se zabývá zpracováním řečového signálu. Aby bylo možné provádět rozpoznávání řeči musí být řeč nejdříve převedena do formátu vhodného pro tuto úlohu. Je nutné provést předzpracování a získat z řeči parametry vhodné pro systémy rozpoznávání řeči. Jsou zde popsány některé metody, kterými lze tyto parametry získat. Následující kapitola se zabývá metodami pro klasifikaci a popisuje ty nejpoužívanější. Budu zde popisovat obecně strojové učení, rozdělení systémů pro rozpoznávání řeči a některé z algoritmů, které se k tomu používají.

Práce pokračuje kapitolou 5, která se věnuje konkrétním nástrojům, které budou použity při vytváření systému. Na základě rešerše stávajících systémů jsem vybrala pro extrakci příznaků MFCC (Mel-frequency Cepstral Coefficients) a pro klasifikaci konvoluční neuronovou síť. V této kapitole jsou popsány nástroje Keras a TensorFlow, které byly při realizaci použity a datová sada. V poslední kapitole je implementace navrženého systému. Je zde popsán princip fungování kódu a také výsledky, kterých se podařilo dosáhnout. Výstupem je systém pro automatické rozpoznávání izolovaných slov v programovacím jazyce Python.

1 Rešerše systémů využívajících automatické rozpoznávání řeči

Tato kapitola uvádí systémy, které využívají ASR a jejich popis. Z toho dále vyplyne, jaké jsou možnosti pro realizaci takovýchto systémů, které budou podrobněji popsány v dalších kapitolách.

V literatuře [1] se autoři zaměřili na rozpoznávání izolovaných slov v bezšumovém prostředí. Pro parametrizaci řeči jsou použity MFCC. Pro samotné zpracování řeči je zde použita metoda DTW (Dynamic Time Warping) a pro klasifikaci je použito KNN (K-Nearest Neighbors). Systém byl testován pro rozpoznávání izolovaných slov v angličtině na malém slovníku. Experiment probíhal s pěti řečníky v akusticky vyváženém, bezšumovém prostředí. Implementace rozpoznávače řeči byla realizována pomocí softwaru MATLAB R2014b. Autoři prezentovali výsledky pomocí matice záměn (confusion matrix), přesnost rozpoznávání experimentu byla 98.4 % s chybovostí 1.4 %. Algoritmus DTW byl použit také ve [2] a [3].

V publikaci [4] jsou popsány metody pro rozpoznávání izolovaných slov za použití MFCC pro parametrizaci a klasifikátoru vektorové kvantizace (VQ – Vector Quantization). Tento systém je navržen pro univerzitního telefonního operátora, má za úkol zjistit, jakým jazykem chce volající mluvit, z jaké je fakulty a poté volajícímu umožní komunikovat s příslušnými pracovníky univerzity. Dále je v publikaci popsáno samotné zpravování řeči a princip vektorové kvantizace. Trénování a testování probíhalo ve třech iteracích z nichž první měla velikost kódové knihy 27 vektorů a úspěšnost rozpoznávání 31.11 %, druhá měla velikost kódové knihy 54 vektorů a úspěšnost $85,71$ % a třetí verze měla 81 vektorů a přesnost rozpoznání 88.88 %. Pro testování bylo nasbíráno 100 nahrávek od různých řečníků s různou úrovní šumu. Z výsledků autoři usuzují, že při zvětšování kódové knihy roste přesnost rozpoznávání.

V [6], [7] a [8] bylo pro klasifikaci použito HMM (Hidden Markov Model) a dále pak autoři [5] vytvořili systém pro rozpoznávání izolovaných slov založený na HMM a DNN (Deep Neural Network). Je zde použita vícevrstvá neuronová síť s dopředným šířením, která vytváří pravděpodobnosti pro stavy HMM, vstupem sítě je vektor příznaků. Dále autoři použili síť typu DBN (Deep Belief Network) pro trénování vícevrstvé neuronové sítě, aby dosáhli lepšího výkonu. Vektor příznaků je tvořen MFCC koeficienty. Výsledky byly vyhodnoceny na databázi TI digits – databáze obsahující číslice nula až devět. Trénování a testování probíhalo s nahrávkami 55 mužů a 57 žen. Byla zjišťována přesnost pro různé počty skrytých vrstev, pro 300 skrytých vrstev byla přesnost 83.76%, pro 400 skrytých vrstev byla přesnost 85.44% a pro 500 skrytých vrstev byla přesnost 86.06%.

Autoři v práci [9] se zaměřují na automatické rozpoznávání propojených slov v anglickém a hindském jazyce. Rozpoznávání propojených slov je vytvořeno jako rozšíření systému pro rozpoznávání izolovaných slov, který je závislý na řečníkovi. Při předzpracování byl kladen důraz na redukování šumu a pro rozdělení vět na slova byl modifikován VAD (Voice Activity Detector), parametrizace pak probíhala metodou MFCC. Byla vytvořena databáze izolovaných slov. Trénovací sada obsahovala 20 slov od 10 řečníků. Ve fázi testování bylo provedeno 5 pozorování, které se skládaly z 5 anglických a hindských vět. Autoři v práci porovnávají klasifikátory DTW

a HMM. Výsledky ukazují, že metoda HMM má pro angličtinu menší chybovost než metoda DTW. Hindština vykazovala stejné průměrné chybovosti u obou metod.

T. Sainath a spol. [10] se zabývají použitím CNN (Convolutional Neural Network) pro vyhledávání klíčových slov (KWS – KeyWord Spotting), jsou zde vyzdviženy výhody CNN oproti DNN (Deep Neural Network) – zlepšení výkonu a zmenšení velikosti modelu. Autoři předpokládali dvě různé aplikace. První uvažuje omezený počet výpočtů KWS systému – toto omezení nedovoluje typické architektury, které fungují dobře pro CNN ve frekvenční oblasti, proto autoři navrhli novou architekturu, která tato výpočetní omezení respektuje. Druhá aplikace uvažuje limitování počtu parametrů a je realizována podvzorkováváním v časové a frekvenční oblasti. Autoři porovnávali CNN a DNN na 14 různých frázích při omezeném počtu parametrů, datová sada obsahovala přibližně 10 až 15 tisíc promluv těchto frází a 396 tisíc jiných promluv. Výsledkem tohoto srovnání autoři zjistili, že při posouvání konvolučních filtrů ve frekvenční oblasti bylo relativní zlepšení výkonu přes 27% oproti DNN v čistých i zašuměných podmínkách. Při omezování parametrů bylo relativní zlepšení přes 41%.

V literatuře [11] se autoři zaměřili na vyhledávání klíčových slov s malým záznamem v paměti, nízkou výpočetní cenu a vysokou přesnost a navrhli systém založený na hlubokých neuronových sítích. Tento systém porovnávali s běžně užívanou technikou HMM pro klasifikaci a byl zaměřen na výrazně jednodušší implementaci. Autoři použili dopřednou plně propojenou neuronovou síť s k skrytými vrstvami a n skrytými uzly pro každou vrstvu a ReLU (Rectified Linear Units) aktivační funkce. Experimentální výsledky ukázaly, že navržený systém překonával standardní HMM v zašuměných i nezašuměných podmínkách.

2 Zpracování řeči

Pro většinu lidí je řeč nejjednodušší způsob komunikace, avšak jako signál je řeč velmi komplexní. Je to akustický signál s mnoha úrovněmi informací jako například fonémy, slabiky, slova a věty. Řeč kromě obsahového významu přenáší také informace o mluvčím a prostředí, pro automatické rozpoznávání řeči může toto komplikovat dekódování přenášeného signálu. Pro vytvoření systému rozpoznávání řeči je potřeba definovat matematický model řeči, toto je velmi složité kvůli komplexnosti řečového signálu.

Zpracování řeči je multidisciplinární obor, kterým se zabývají přírodní, humanitní i technické vědy. Příkladem může být zpracování signálů, fyziologie, akustika, fonetika, fonologie a mnoho dalších. Těmto vědám odpovídají také úrovně zpracování. V této kapitole popíšu především to, co je důležité pro rozpoznávání řeči.

Nejmenší řečová jednotka z hlediska fonetiky je hláska. Z hlediska fonologie je to foném a ten určuje nejmenší významovou jednotku. Každý jazyk obsahuje jiné fonémy, například čeština jich má 36. Dále je možné dělit řečové jednotky v českém jazyce například na samohlásky, souhlásky a slabiky.

Důležitým pojmem jsou také tzv. formantové frekvence, což jsou frekvence, okolo kterých se soustředí akustické energie a jsou určeny artikulačním traktem (viz. kapitola 2.1). Formanty jsou důležité například pro syntézu řeči.

2.1 Tvorba řeči

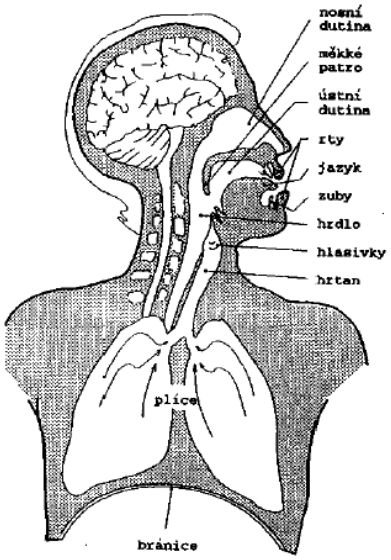
Řeč se v lidském těle vytváří v tzv. hlasovém traktu (obr. 2.1), který je složen z řečových orgánů. Hlasový trakt se skládá ze tří základních ústrojí. Prvním je dechové ústrojí, které je základním zdrojem energie (dýchací cesty, plíce). Druhá část je hlasové ústrojí, jehož nejdůležitější částí jsou hlasivky. Kmitáním hlasivek se tvoří základní hlasivkový tón, který je více popsán v kapitole 3.2, při kmitání hlasivek se tvoří znělé zvuky a v klidovém stavu hlasivek jsou vytvářeny neznělé zvuky (šum). Poslední částí hlasového traktu je artikulační ústrojí, které má za úkol například vytváření tónové struktury, nebo šumové složky.

Matematický model pak obsahuje zdroj energie, modulaci energie (v číslicovém zpracování buzení) a modifikující ústrojí (filtrace), toto je naznačeno na obrázku 2.2

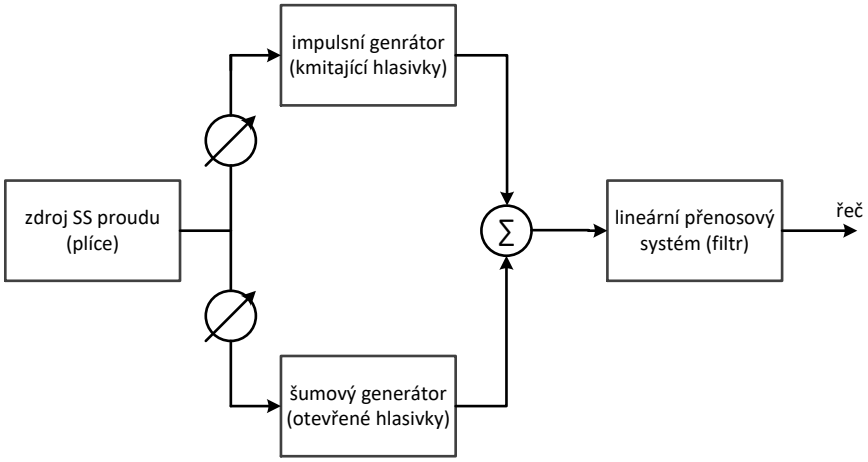
Z číslicového modelu na obrázku 2.2 vyplývá, že řeč je dána konvolucí budící funkce $g(t)$ a impulzní odezvy $h(t)$ hlasového traktu (vztah 1), což ve frekvenční oblasti odpovídá násobení spekter buzení $G(f)$ a impulzní odezvy $H(f)$ (vztah 2)

$$s(t) = g(t) \star h(t) = \int_{-\infty}^{\infty} g(\tau)h(t-\tau)d\tau, \tag{1}$$

$$S(f) = G(f)H(f). \tag{2}$$



Obrázek 2.1: Hlasový trakt [13]



Obrázek 2.2: Číslicový model hlasového traktu

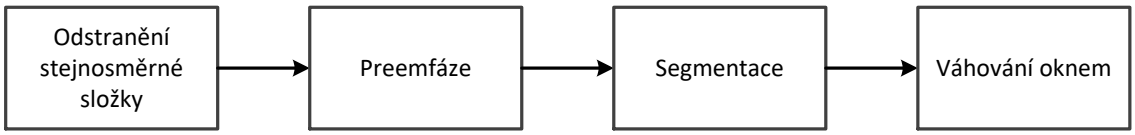
3 Zpracování řečového signálu

Zpracování se provádí, protože řečový signál má nestacionární charakter a některé jeho vlastnosti nejsou vhodné pro parametrizaci. Zpracování řečového signálu se skládá z předzpracování a extrakce příznaků.

Řeč je signál, u kterého se předpokládá, že se v čase pomalu mění – je krátkodobě stacionární. Z toho důvodu se řeč během předzpracování rozděluje na krátké úseky (segmenty, rámce), na kterých se provádí extrakce příznaků. Tyto příznaky se pak používají v klasifikátorech pro samotné rozpoznávání.

3.1 Předzpracování řečového signálu

Předzpracování má za úkol připravit signál pro extrakci příznaků, tento proces je znázorněn na obrázku 3.1.



Obrázek 3.1: Předzpracování řeči

Odstranění stejnosměrné složky

Stejnosměrná (SS) složka je součást signálu, která může negativně ovlivňovat další zpracování a výpočty, proto je třeba ji z řeči odstranit. SS složku lze spočítat jako střední hodnotu ze všech analyzovaných vzorků a ze signálu se pak odstraní jejím odečtením.

Výpočet SS složky:

$$\mu_s = \frac{1}{N} \sum_{n=1}^{N} s[n]. \tag{3}$$

Odečtení SS složky od řečového signálu:

$$s'[n] = s[n] - \mu_s. \tag{4}$$

SS složku lze odstranit v reálném čase, pokud není znám celý průběh signálu (např. při snímání z mikrofonu) na základě střední hodnoty předchozího vzorku. Předchozí a aktuální vzorek jsou spojeny konstantou γ, jejíž hodnota se blíží 1. Vzorec 5 ukazuje výpočet SS složky aktuálního vzorku [14]. SS složka je odstraněna odečtením střední hodnoty podobně jako v rovnici 4.

$$\mu_s[n] = \gamma\mu_s[n-1] + (1-\gamma)s[n], \tag{5}$$

kde $\mu_s[n-1]$ je střední hodnota předchozího vzorku, $\mu_s[n]$ je střední hodnota aktuálního vzorku a $s[n]$ je skutečná hodnota aktuálního vzorku.

Preemfáze

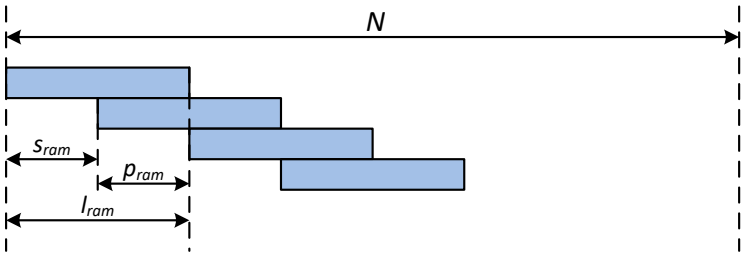
Preemfáze se provádí z důvodů vyrovnání frekvenční charakteristiky a má za úkol kompenzovat útlum energie signálu ve vyšších frekvencích. Provádí se aplikováním jednoduchého FIR (Finite Impulse Response) filtru prvního řádu, kde filtrace je dána vztahem:

$$s''[n] = s'[n] - ks'[n-1], \tag{6}$$

pro parametr k se volí hodnoty od 0.9 do 1.

Segmentace

Řeč je obecně nestacionární signál, který není možné zpracovávat zde popisovanými metodami. Předpokládá se ale, že řeč je krátkodobě stacionární, a proto ji rozdělujeme na krátké úseky, na kterých můžeme provádět analýzu. Tyto úseky (rámce) jsou obvykle dlouhé 20 až 30 ms s překryvem z důvodu vyhlazení časové změny parametrů (viz. obrázek 3.2). Délka překrytí se obvykle volí 10 až 15 ms (polovina segmentu).



Obrázek 3.2: Předzpracování řeči

Posun rámce s_{ram} se vypočte dle vztahu 7, kde l_{ram} je délka rámce a p_{ram} je délka překrytí

$$s_{ram} = l_{ram} - p_{ram}. \tag{7}$$

Počet rámců se dále vypočte:

$$N_{ram} = 1 + \left\lfloor \frac{N - l_{ram}}{s_{ram}} \right\rfloor, \tag{8}$$

kde N_{ram} je celkový počet rámců a N je délka signálu

Váhovací funkce

Váhovací funkce se používá pro odstranění ostrých přechodů mezi jednotlivými rámci, které mohou vzniknout při segmentaci a mohou dále působit problémy při analýze signálu. Nejčastěji se používá pravoúhlé (přiřazuje všem vzorkům stejnou váhu) nebo Hammingovo (potlačuje krajní hodnoty) okénko. Matematické definice těchto váhovacích funkcí jsou následující:

Pravoúhlé okno

$$w[n] = \begin{cases} 1 & \text{pro} \quad 0 \leq n \leq L-1, \\ 0 & \text{pro} \quad \text{ostatní } n. \end{cases} \tag{9}$$

Hammingovo okno:

$$w[n] = \begin{cases} 0.54 - 0.46 \cos\left(\frac{2\pi n}{L-1}\right) & \text{pro} \quad 0 \leq n \leq L-1, \\ 0 & \text{pro} \quad \text{ostatní } n, \end{cases} \tag{10}$$

kde L je počet vzorků, vybraných okénkem.

Z definice okenních funkcí můžeme vypozorovat, že kromě přesně definované části (délky jednoho rámce) je zbytek signálu utlumen. Během zpracování řečového signálu se okénko posouvá o přesně stanovený krok a další charakteristiky se počítají právě na jednotlivých úsecích (rámcích) signálu, obvykle se volí délka rámce tak, aby odpovídala 10 až 40 ms řeči – na takto krátkých úsecích lze pozorovat změny, které jsou důležité pro analýzu řeči [13].

3.2 Charakteristiky řeči

Charakteristiky řeči mohou být skalární - jednočíselné, mezi tyto charakteristiky patří krátkodobá energie, počet průchodu nulou a další. Další charakteristiky jsou vektorové - jejich výsledkem je řada čísel (vektor) popisující řečový signál, mezi tyto charakteristiky lze zařadit například LPC (Linear Predictive Coding) nebo MFCC.

3.2.1 Skalární charakteristiky

Krátkodobá energie

$$E_n = \sum_{k=-\infty}^{\infty} \left(s[k]w[n-k]\right)^2. \tag{11}$$

Rovnice 11 uvádí vztah pro krátkodobou energii, kde $s[k]$ je vzorek signálu v čase k a $w[n]$ je okenní funkce. Kvůli velké citlivosti na změny signálu způsobené vlivem kvadrátu je možno

zavést ještě vztah pro krátkodobou intenzitu:

$$M_n = \sum_{k=-\infty}^{\infty} |s[k]|\, w[n-k]. \tag{12}$$

Tyto vtahy lze využít například pro oddělení znělých a neznělých úseků řeči nebo pro oddělení ticha od segmentů řeči. Vzhledem k tomu že zde dochází ke znehodnocení informačního obsahu řeči nelze krátkodobé charakteristiky použít pro rekonstrukci. Energetické charakteristiky zachycují změny signálu v čase a mohou být použity pro doplnění ostatních charakteristik.

Počet průchodu nulou

Funkce počtu průchodu nulou zjišťuje změny polarity signálu. Podle toho se dají například rozlišit znělé hlásky od neznělých.

$$Z_n = \sum_{n=1}^{l_{ram}} \left|\operatorname{sgn}\left(s[n]\right) - \operatorname{sgn}\left(s[n-1]\right)\right|, \tag{13}$$

kde sgn je znaménková funkce:

$$\operatorname{sgn}\left(s[n]\right) = \begin{cases} 1 & \text{pro} \quad s[n] \geq 0, \\ -1 & \text{pro} \quad s[n] < 0. \end{cases} \tag{14}$$

Základní hlasivkový tón

Základní hlasivkový tón nebo také základní frekvence řeči určuje výšku hlasu řečníka. Tento tón je určen hlasovým ústrojím každého člověka a liší se podle pohlaví a může být také ovlivněn emočním stavem řečníka, věkem či jinými vlivy. Rozsah základní frekvence bývá přibližně 50-400 Hz [13] (u můžů to je přibližně 80-160 Hz, u žen 150-300 Hz a u dětí 200-500 Hz). Tento parametr lze určit jak v časové tak i ve frekvenční oblasti.

3.2.2 Vektorové charakteristiky

Lineární prediktivní kódování

Lineární prediktivní kódování (LPC) je metoda pro analýzu zvukového signálu, která odhaduje parametry řeči tak, že předpokládá, že každý vzorek signálu se dá popsat lineární kombinací předchozích vzorků a budící funkce. Pomocí LPC lze vypočíst vyhlazený odhad spektrální výkonové hustoty a také jej lze použít pro formantovou analýzu řeči.

Výpočet LPC vychází z homomorfního zpracování řeči. Model vytváření řeči lze popsat přenosovou funkcí:

$$H(z) = \frac{G}{A(z)} = \frac{G}{1 + \sum_{i=1}^{Q} a_i z^{-i}}, \tag{15}$$

kde G je zesílení a Q je řád modelu. Kepstrální koeficienty pak lze vypočíst pomocí vztahů:

$$c(1) = -a_1, \tag{16}$$

$$c(k) = -a_k - \sum_{i=1}^{k-1} \left(\frac{i}{k}\right) c(i) a_{k-1}, \; pro \; 2 \leq k \leq Q, \tag{17}$$

$$c(k) = \sum_{i=1}^{Q} \left(\frac{k-i}{k}\right) c(k-i) a_i, \; pro \; k \; > Q. \tag{18}$$

$$\tag{19}$$

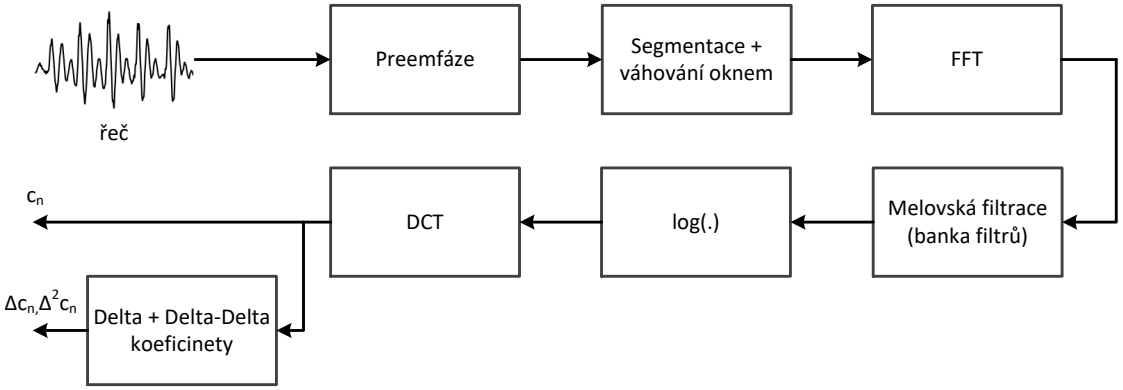
Melovské kepstrální koeficienty

Melovské kepstrální koeficienty (MFCC) se snaží do jisté míry respektovat nelineární vnímání zvuku lidským uchem – na nižších frekvencích je vetší rozlišení než na vysokých, toto lze využít při rozpoznávání řeči. Nelinearita je kompenzována využitím trojúhelníkových pásmových filtrů s lineárním rozložením frekvencí na tzv. melovské frekvenční škále, která je definována vztahem:

$$f_m = 2595 \log_{10} \left(1 + \frac{f}{700}\right), \tag{20}$$

kde f[Hz] je frekvence v lineární škále a f_m[mel] je odpovídající frekvence v nelineární melovské stupnici [13].

Blokové schéma výpočtu MFCC je na obrázku 3.3. Ze schématu je předzpracování popsáno v kapitole 3.1, ostatní je popsáno níže.



Obrázek 3.3: Výpočet MFCC

Rychlá Fourierova transformace

Výpočet rychlé Fourierovy transformace (FFT – Fast Fourier Transform) vyžaduje délku signálu 2^n, což je možno udělat při váhovaní oknem, nebo signál na požadovanou délku doplnit nulami.

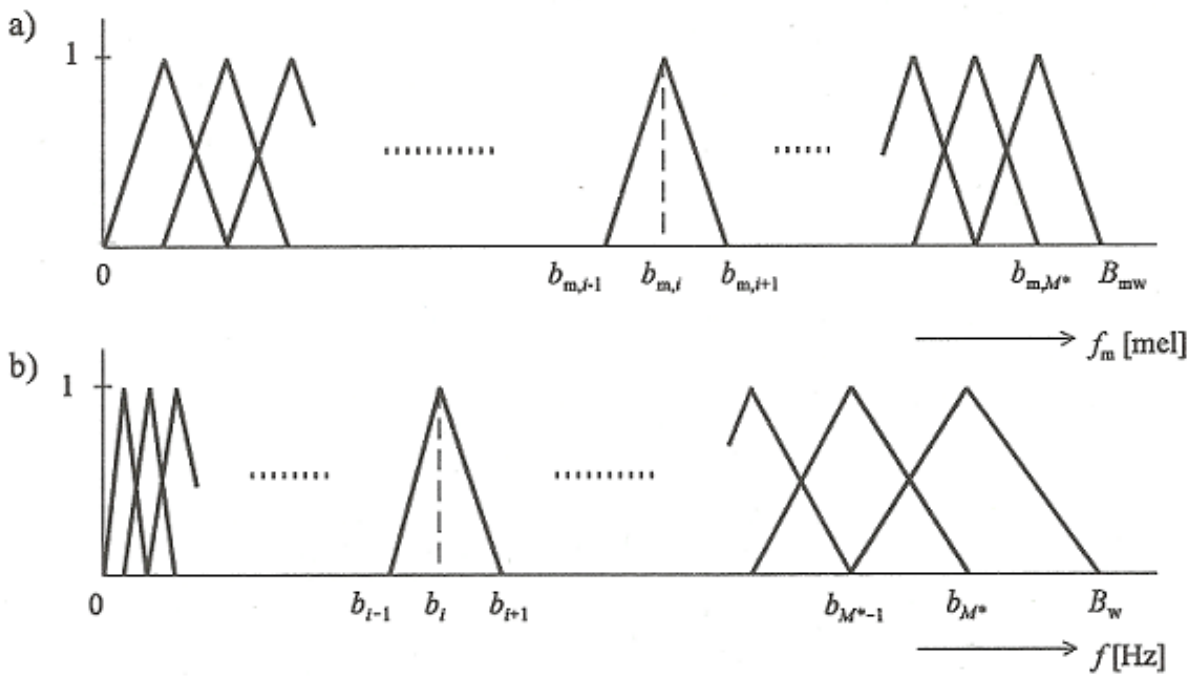
Výsledkem je amplitudové spektrum jednotlivých rámců signálu.

Melovská filtrace

V dalším kroku se na každý rámec aplikuje banka trojúhelníkových filtrů s rovnoměrným rozložením podél frekvenční osy v melovské škále (obrázek 3.4) – filtry jsou rozloženy po celém frekvenčním pásmu od nuly až do poloviny vzorkovací frekvence. Počet pásem se volí dle vzorkovací frekvence a zároveň je třeba respektovat vzorkovací teorém. Běžně používané hodnoty počtu pásem jsou v tabulce 3.1 [13].

Tabulka 3.1: Typické hodnoty počtu filtrů M^* pro dané přenášené pásmo [13].

Frekvence vzorkování f_vz [Hz]	8000	11000	16000	22000	44000
Přenášené pásmo (0;B_w) [Hz]	(0;4000)	(0;5500)	(0;8000)	(0;11000)	(0;22000)
Přenášené pásmo (0;B_{mw}) [mel]	(0;2146)	(0;2458)	(0;2840)	(0;3174)	(0;3921)
Počet pásem M^*	15	17	20	22	27



Obrázek 3.4: Banka trojúhelníkových filtrů: a) v melovské škále, b) v původní frekvenční škále[13].

Výpočet logaritmů

Na výstup z filtrace se dále aplikuje přirozený logaritmus, který omezuje dynamiku signálu a je předpokladem pro případné další zpracování.

Diskrétní kosinová transformace (DCT)

Dále je třeba provést zpětnou Fourierovu transformaci, kterou je možno redukovat na DCT, protože výkonové spektrum je reálné a symetrické.

$$c_m(j) = \sum_{i=1}^{M^*} \log y_m(i) \cos\left(\frac{\pi j}{M^*}(i - 0,5)\right), \ pro\ j = 0, 1, ..., M, \tag{21}$$

kde $y_m(i)$ je odezva filtru, M^* je počet pásem melovského pásmového filtru a M je počet melovských kepstrálních koeficientů [13].

Delta a Delta-Delta koeficienty

Delta a Delta-Delta koeficienty jsou tzv. dynamické koeficienty, které se vypočítají z několika rámců a vyjadřují časové změny pro každý analyzovaný rámec.

MFCC s delta a delta-delta koeficienty jsou nekorelované a mají vysoký informační obsah, proto jsou vhodné pro použití zejména v aplikacích automatického rozpoznávání řeči.

4 Automatické rozpoznávání řeči

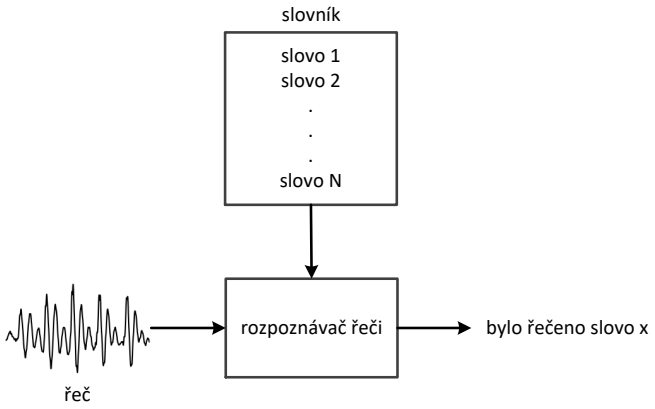
Automatické rozpoznávání řeči je obtížná úloha z několika důvodů. Hlas každého člověka je jiný a liší se od ostatních. Řeč se také mění v různých situacích dle emočního stavu řečníka. ASR může být závislé na řečníkovi, případně složitější systémy jsou nezávislé na řečníkovi. Dalším problémem může být okolní rušení a šum pozadí, který může rozpoznávání ovlivňovat. Základní rozdělení ASR dle složitosti konstrukce rozpoznávače:

- rozpoznávání izolovaných slov
- rozpoznávání spojených slov
- rozpoznávání plynulé řeči

Princip rozpoznávání izolovaných slov je naznačen na obrázku 4.1.

Dále lze rozpoznávání rozdělit do dvou hlavních skupin dle metod rozpoznávání. Jsou to metody, které pracují na principu porovnávání s referencemi jako je například metoda dynamického borcení času – tyto metody se používaly v minulosti a to zejména pro úlohy rozpoznávání izolovaných slov. Druhou skupinou klasifikátorů jsou statistické metody, kde je možno modelovat celá slova nebo části slov (slabiky, fonémy atd.) promluvy pomocí tzv. skrytých Markovových modelů.

Jedním z novějších přístupů ve vytváření systémů pro ASR jsou umělé neuronové sítě (ANN – Artificial Neural Network). Existuje mnoho různých způsobů, jak vytvořit rozpoznávač řeči s neuronovými sítěmi. Příkladem může být kombinace ANN a HMM, kde ANN vytvářejí rozložení pravděpodobností pro jednotlivé stavy HMM. Dále existují rozpoznávače využívající vícevrstvé, rekurentní nebo konvoluční neuronové sítě (některé možnosti jsou zmíněny v kapitole 1).



Obrázek 4.1: Princip rozpoznávání izolovaných slov

Strojové učení

Strojové učení je obor, který se zbývá metodami identifikace a implementace systémů a algoritmů, kterými se může počítač učit v závislosti na daných příkladech a vstupu. Výzva strojového učení spočívá v tom, aby se počítač naučil automaticky rozpoznávat komplexní vzory a dělal co možná nejlepší rozhodnutí. Proces učení vyžaduje data, která jsou rozdělena na trénovací sadu a testovací sadu. Trénovací sada jsou data, na kterých se model učí, během této fáze mohou být různě upravovány parametry modelu tak, aby odpovídaly požadavkům. Testovací sada pak slouží k vyhodnocení výkonu modelu na datech, která nezná [15].

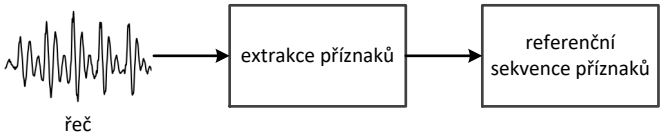
Strojové učení lze rozdělit podle způsobu učení na 3 druhy:

- učení s učitelem
- učení bez učitele
- zpětnovazební učení

4.1 Metoda dynamického borcení času

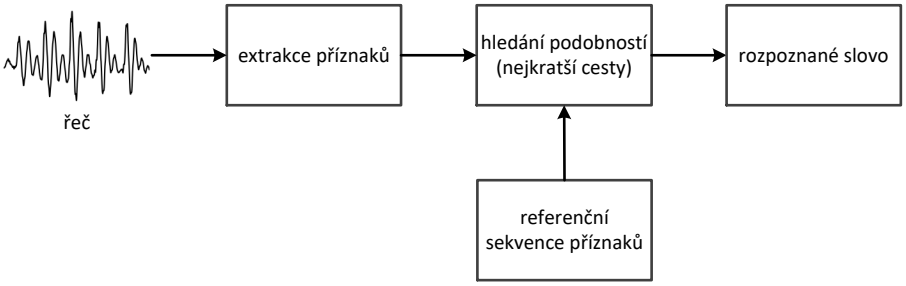
Metoda DTW je založená na přímém srovnávání slov s referencemi nebo také počítání vzdáleností mezi řečníkem pronesenými slovy a slovy, které jsou uložené ve slovníku (princip je naznačen na obrázku 4.3). Slova mohou mít různou variabilitu v čase i prostoru. Tato metoda se používá pro rozpoznávání izolovaných slov s malým slovníkem – u velkých slovníků vzniká problém s velkými nároky na výpočetní výkon a paměť. DTW se používá v systémech závislých na řečníkovi s malými slovníky. Pro extrakci příznaků je možno použít například MFCC.

Tuto metodu je možné logicky rozdělit do dvou fází. V první fázi se vytvoří referenční matice vzorů pro jednotlivá slova v databázi (obrázek 4.2). V druhé fázi se po provedení extrakce příznaků z řeči hledá nejkratší cesta mezi referenční a testovací sekvencí příznaků (obrázek 4.3).



Obrázek 4.2: DTW – první fáze (trénování)

Při hledání nejlepší shody mezi referenční $r(k)$ a testovací $t(k)$ sekvencí příznaků můžeme najít cestu mřížkou, kde k je časová proměnná (reference může být na svislé ose a testovací sekvence na vodorovné osy mřížky), která minimalizuje celkové vzdálenosti mezi těmito sekvencemi. Jakmile je nalezena nejkratší cesta pro tuto referenci je uložena a počítá se pro ostatní



Obrázek 4.3: DTW - druhá fáze (rozpoznávání)

vzory (reference). Takto se hledá vzdálenost mezi všemi možnými cestami mřížkou a pro každou cestu se počítá celková vzdálenost – z tohoto lze usuzovat, že počet možných cest mřížkou bude velmi velký a výpočetně náročný, z tohoto důvodu se provádí optimalizace výpočtu.

Optimalizace dle [16] je možná:

1. Podmínka monotónnosti – cesta se nebude vracet, indexy referenčních i testovacích sekvencí budou vždy růst, nebo zůstanou stejné (nebudou klesat).
2. Podmínka kontinuity – cesta bude postupovat postupně po jednom kroku v referenční i testovací sekvenci.
3. Mezní podmínka – cesta bude začínat v levém dolním rohu a končit v pravém horním rohu.
4. Přizpůsobení délce okna – maximální vzdálenost, kterou může cesta projít je délka okna.
5. Omezení sklonu – přechody by neměly být příliš jemné ani příliš strmé.

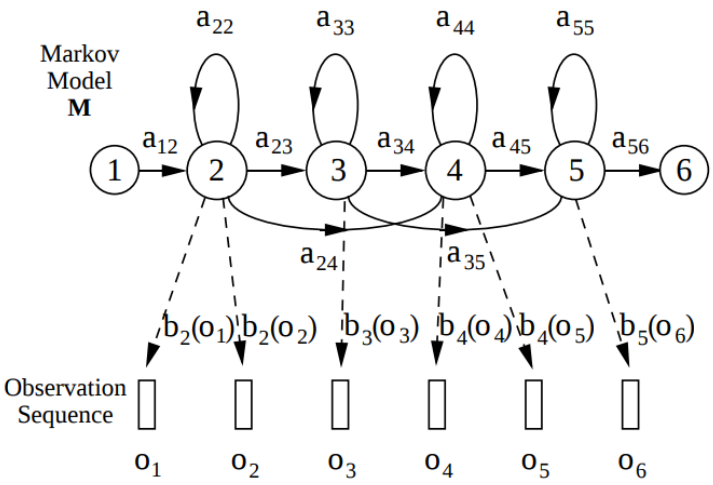
Aplikace těchto podmínek by měla snížit počet cest, které je potřeba brát v úvahu při hledání nejkratší cesty.

Pro normalizaci délky cesty se používá váhovací funkce, která může být symetrická nebo asymetrická.

4.2 Skryté Markovovy modely

Základní předpoklad statistického modelování je, že signál je náhodný a je ho možné charakterizovat parametrickým náhodným procesem. Tyto parametry pak lze přesně stanovit. Skryté Markovovy modely jsou zdroje pravděpodobnostní funkce pro Markovovy řetězce [17].

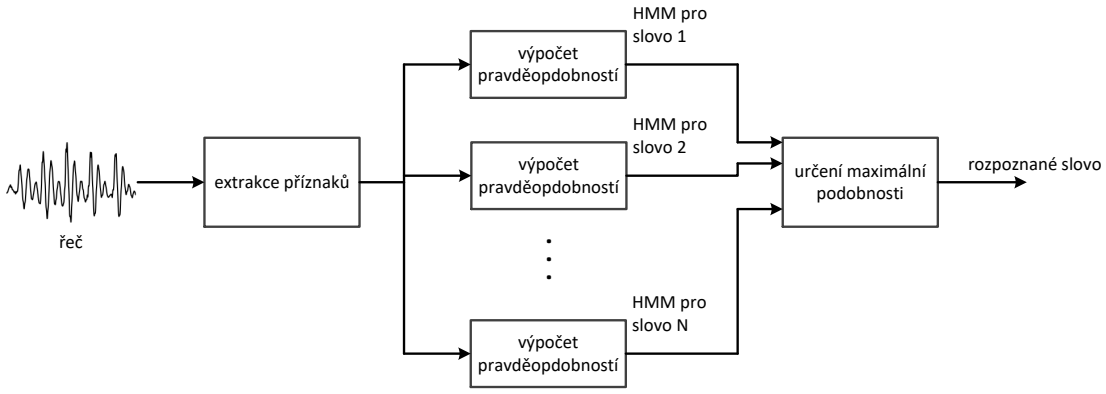
Skrytý Markovův model je model statistického procesu, který je reprezentován pravděpodobnostním konečným stavovým automatem. Struktura HMM pro ASR jsou obvykle levo-pravé Markovovy modely, příklad takového modelu je na obrázku 4.4.



Obrázek 4.4: Příklad struktury HMM [18]

Pro rozpoznávání izolovaných slov můžeme předpokládat, že máme slovník o velikosti N slov a každé slovo je modelováno jiným HMM. Předpokládejme, že pro každé slovo máme trénovací sadu, která obsahuje K výskytů každého proneseného slova a každý výskyt má sekvenci příznaků. Pro rozpoznávání izolovaných slov je nejprve potřeba vytvořit HMM model pro každé slovo ve slovníku a odhadnout jeho parametry. Dále je potřeba pro každé neznámé slovo provést analýzu příznaků a vypočíst maximální podobnost s modely, které jsou ve slovníku (obrázek 4.5). Určení maximální pravděpodobnosti může být provedeno například Viterbiho algoritmem.

Systém rozpoznávání plynulé řeči je obvykle složitější a jeho struktura obsahuje kromě akustického modelování obsahuje také lexikální dekódování, syntaktickou a sémantickou analýzu.



Obrázek 4.5: Blokové schéma pro rozpoznávání izolovaných slov pomocí HMM

4.3 Hluboké neuronové sítě

Hluboké učení je část strojového učení založena na konkrétním typu učícího algoritmu, který je charakterizován snahou vytvořit model učení s několika úrovněmi z nichž nejhlubší úrovně mají jako vstup výstupy předchozích úrovní. Tento pohled na vrstvení úrovní učení je inspirován způsobem jakým lidský mozek zpracovává informace a učí se reagovat na vnější podněty.

Hluboké neuronové sítě jsou neuronové sítě, které mají komplexnější modely s velkými počty neuronů, skrytých vrstev a spojení. Jsou aplikovatelné na problémy strojového učení s učitelem (supervised learning). DNN pracují paralelně tak, aby mohly zpracovávat velké množství dat a vyžadují trénovaní, které správně určí váhy spojení jednotlivých neuronů.

4.3.1 Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou primárně používány pro zpracování obrazu, ale je možné je použít také pro jiný typ vstupních dat jako je například zpracování zvuku. Jednou z možností použití pro rozpoznávání řeči je analýza spektrogramu. Místo plně propojených skrytých vrstev, které jsou použity ve standardních DNN, jsou zde použity speciální vrstvy tzv. konvoluční a pooling vrstvy.

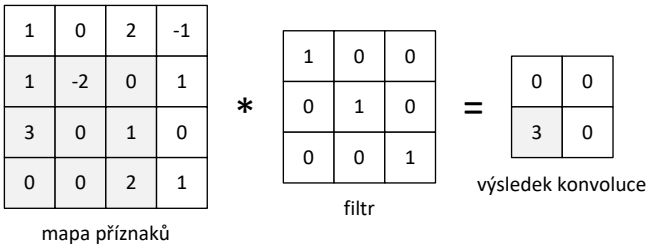
Aby bylo možné použít CNN pro rozpoznávání vzorů vstupní data musejí být organizována do map příznaků. Tento název je odvozen od aplikací zpracování obrazu, kde jsou vstupní data organizovány do dvourozměrné matice pixelů. Pro barevné obrázky pak mohou být hodnoty RGB zobrazeny jako 3 různé mapy příznaků, z nichž každá označuje jednu barvu. CNN posouvá malé okno (filtr) po vstupním obrazu pro trénování i testování tak, aby se váhy uvnitř sítě, které používají tento filtr mohly učit z různých příznaků vstupních dat nezávisle na jejich absolutní pozici [22].

Architektura CNN se obvykle skládá z konvolučních vrstev, aktivačních ReLU vrstev, pooling vrstev a jako poslední by vždy měla být aspoň jedna plně propojená vrstva.

Jednou z populárních metod je použití spektrogramu jako vstup pro CNN. V tomto případě se konvoluční filtry aplikují na dvourozměrný obraz spektrogramu. Toto využití CNN je popsáno v [19], [20] a [21].

Konvoluční vrstvy

Tato vrstva provádí operaci konvoluce na mapách příznaků pomocí tzv. filtrů. Je zde třeba určit velikost filtru a posun. Na vstupu je matice příznaků, na kterou jsou aplikovány sady filtrů. Každý filtr je posouván po matici a počítá se produkt filtru a vstupních dat. Tento proces generuje mapu příznaků pro různé další filtry. Na obrázku 4.6 je naznačen princip výpočtu konvoluce při velikosti mapy příznaků 4x4 a velikosti filtru 3x3.



Obrázek 4.6: Princip konvoluční vrstvy [23]

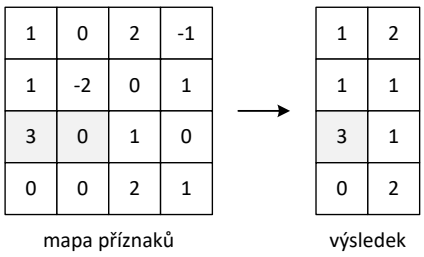
ReLU vrstvy

Po každé konvoluční vrstvě by měla následovat tzv. aktivační ReLU vrstva. Důvodem této vrstvy přivést nelinearitu do systému, který během konvolučních vrstev počítá pouze s lineárními operacemi. Aktivační funkce je obvykle $f(x) = max(0, x)$. Tato vrstva také pomáhá zmírnit problém, kdy trénování spodních vrstev modelu je velmi pomalé oproti vyšším vrstvám.

Pooling vrstvy

Pooling vrstvy zmenšují velikost map příznaků, mohou být také označeny jako nelineární podvzorkování. V podstatě se na každou mapu příznaků aplikuje filtr, který má stejnou velikost i posun. Nejobvyklejší je max pooling, což je filtr, který bere vždy maximální hodnotu. Další možnost může být například average pooling, kde je vždy počítána průměrná hodnota. Tyto vrstvy snižují výpočetní náročnost a kontroluje nebezpečí přeučení[1] sítě. Princip pooling vrstvy je na obrázku 4.7, kde je mapa příznaků o velikosti 4x4 a pooling filtr o velikosti 2x1.

Po každé konvoluční vrstvě by měla následovat také pooling vrstva, protože kromě snížení výpočetní náročnosti tato vrstva také zvyšuje robustnost a mění polohu lokálních vzorů [25].



Obrázek 4.7: Příklad funkce pooling vrstvy [23]

[1]Přeučení označuje stav, kdy model dobře rozpoznává trénovací data, ale není schopen dobře rozpoznávat nová data. Tento stav obvykle nasává když má síť přesnost 99-100% na trénovací sadě dat, ale například pouze kolem 50% na testovací sadě.

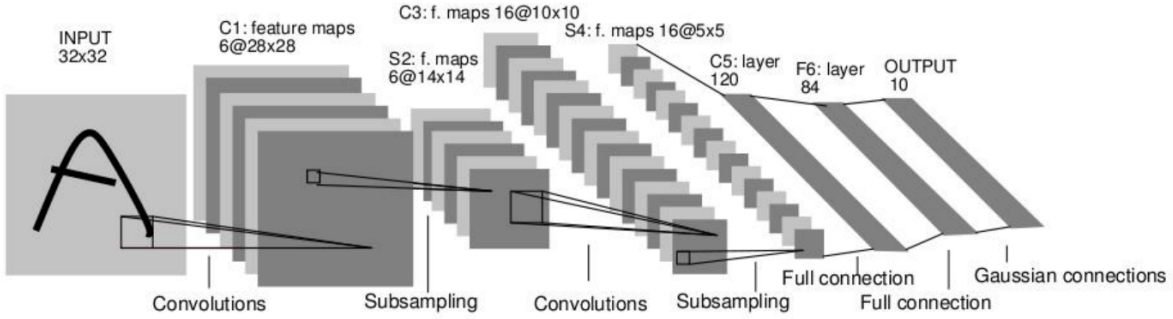
Plně propojená vrstva

Na konci konvoluční sítě je vždy aspoň jedna plně propojená vrstva, která bere vstup z předchozí vrstvy a na výstupu je vždy vektor, jehož velikost odpovídá počtu tříd, ze kterých má systém vybírat. Například pokud bude síť trénovaná pro 10 slov bude výstupem vektor s rozměrem 10.

Dropout vrstvy

Dropout vrstva má za úkol předcházet stavu přeučení. Tato vrstva zahazuje náhodně zvolené váhy mezi neurony na nulu. Síť by pak měla být schopna správně klasifikovat data i přesto, že některá spojení jsou zahozena, zaručuje to aby síť nebyla příliš 'naučená' na trénovací sadu a měla redundantní spojení. Tato vrstva se používá pouze během trénovaní. Vliv dropout vrstev je detailně popsát v [24]

Na obrázku 4.8 je příklad konvoluční sítě LeNet, která byla v [26] použita pro rozpoznávání obrazu, konkrétně pro rozpoznávání rukou psaných číslic.



Obrázek 4.8: Příklad architektury CNN [26]

5 Použité nástroje

V této kapitole budu popisovat nástroje použité pro vytvoření systému rozpoznávání izolovaných slov. Pro vytvoření systému ASR jsem použila programovací jazyk Python.

Extrahované MFCC koeficienty představují vstup pro rozpoznávání. Tyto parametry byly vybrány z důvodu nelineárního vnímání zvuku lidským uchem. Pro klasifikaci je použita konvoluční neuronová síť.

Existuje několik možností pro implementaci neuronových sítí v Pythonu. Je zde řada knihoven a frameworků, které pomáhají vytvářet neuronové sítě. Mezi nejznámější patří Caffe, Theano, TensorFlow a další. Vytvářený systém rozpoznávání řeči je založen na TensorFlow s aplikačním rozhraním Keras. Při práci s daty a signály byly použity převážně knihovny numpy a scipy. Programová a uživatelská dokumentace je generována knihovnou Sphinx

5.1 Keras

Keras je open-source knihovna pro neuronové sítě v Pythonu, která běží nad TensorFlow, Microsoft Cognitive Toolkit nebo Theano. Je navržena pro rychle experimenty s hlubokými neuronovými sítěmi.

Tato knihovna je navržena tak, aby byla uživatelsky přívětivá. Nabízí jednoduché a konzistentní aplikační rozhraní, minimalizuje počet požadovaných uživatelských akcí pro obvyklé případy a poskytuje zpětnou vazbu při chybách v programu.

Výhodou Kerasu je modulárnost. Celý model je chápán jako sekvence nebo graf samostatných, plně konfigurovatelných modulů. Moduly lze libovolně přidávat nebo odebírat, což usnadňuje ladění vytvářených neuronových sítí.

5.2 TensorFlow

TensorFlow je open-source systém pro strojové učení, který používá grafy toku dat pro reprezentaci výpočtů. Výpočty jsou popsány jako graf, který je složen ze sady uzlů. Uzly grafu reprezentují matematické operace, zatímco hrany grafů reprezentují více-dimenzionální pole dat (tenzory) mezi uzly. TensorFlow dovoluje rozdělit výpočty na jeden nebo více procesorů (CPU, nebo GPU). Dále obsahuje nástroj pro vizualizaci dat TensorBoard.

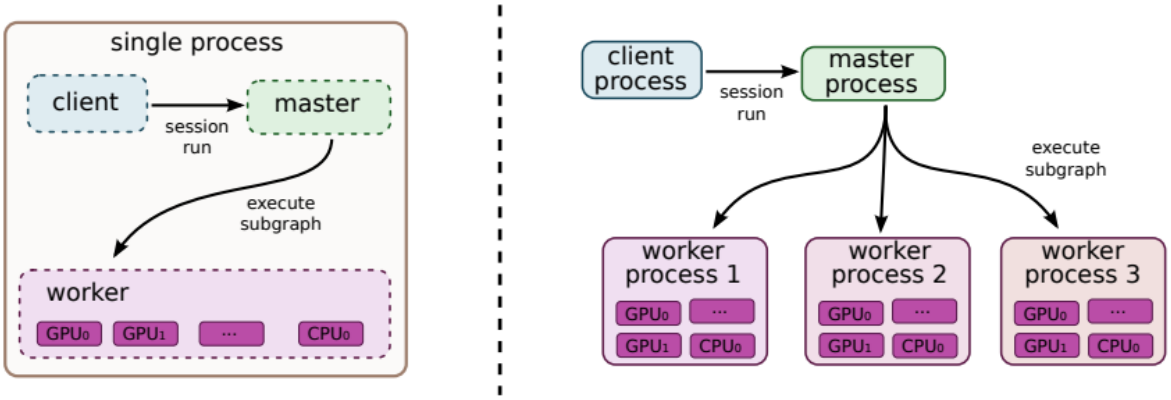
TensorFlow bylo původně vyvíjeno výzkumníky a inženýry pracujícími v Google Brain týmu v Machine Intelligence Research ve společnosti Google, pro účely strojového učení a výzkumu hlubokých neuronových sítí. Systém je použitelný v řadě dalších domén díky tomu, že je dostatečně obecný.

TensorFlow poskytuje stabilní aplikační rozhraní pro Python a jazyk C a aplikační rozhraní, které je zpětně kompatibilní s jazyky C++, Java, Go, JavaScipt a Swift.

Hlavní komponenty TensorFlow jsou klienti (clients), kteří používají rozhraní relace (session) pro komunikaci s nadřazenými (master) procesy. Dále jeden nebo více pracovních procesů, z nichž

je každý zodpovědný za rozhodování o přístupu k jednomu nebo více výpočetním zařízením (jádra CPU, nebo grafické karty), tyto pracovní procesy také provádějí spouštění uzlů grafů na těchto zařízeních dle instrukcí nadřazených procesů. TensorFlow lze implementovat lokálně nebo distribuovaně.

Lokální implementace je použita pokud klientský, nadřazený i pracovní proces běží na jednom zařízení v rámci jednoho procesu operačního systému (případně s více zařízeními, například pokud má zařízení nainstalováno více grafických karet). Distribuovaná implementace sdílí většinu kódu s lokální, ale rozšiřuje ji o podporu prostředí, kde klientský, nadřazený i pracovní proces můžou být v různých procesech na různých zařízeních [27]. Na obrázku 5.1 je znázorněn rozdíl mezi lokální a distribuovanou implementací.



Obrázek 5.1: Lokální a distribuovaná struktura

5.3 Datová sada

Neuronové sítě jsou náročné na velikost trénovacích dat. Pro relativně dobré natrénování a dobrou přesnost rozpoznávání je potřeba mít co největší možnou datovou sadu. Vzhledem k tomu, že není žádná dostupná datová sada česky namluvených slov, zvolila jsem volně dostupnou datovou sadu Speech commands dataset version 2 z [28]. Tato datová sada je navržena tak, aby pomáhala navrhovat, trénovat a vyhodnocovat jednoduché systémy pro rozpoznávání řeči. Primárním cílem je poskytovat možnost sestavit a otestovat malé modely pro detekování izolovaných slov. Tato datová sada je vydána pod licencí Creative Commons BY 4.0. Toto ji umožňuje snadno začlenit do ukázkových programů a dalších skriptů, kde je ji možné stáhnout bez zásahu uživatele.

Tato datová sada obsahuje dohromady 105 829 nahrávek 35 slov od 2618 různých řečníků (žen i mužů) v různých prostředích. Jednotlivá slova a jejich počty jsou vypsány v tabulce 5.1 a jsou složena z příkazů a dalších krátkých slov, která pokrývají mnoho různých fonémů. Nahrávky jednotlivých slov jsou ve formátu wav s kódováním PCM a vzorkovací frekvencí 16 KHz, každý soubor má délku 1 s nebo méně.

Kontrola kvality probíhala tak, že byly nejdříve automaticky odstraněny nahrávky, které byly příliš krátké nebo tiché. Po tomto stále zbyly promluvy, které měly nesprávná slova nebo byly z jiných důvodů nesrozumitelné. Z toho důvodu byla dále kontrola předána komerční firmě. Subjekty měly po poslechu napsat slovo, které slyšely. Každá nahrávka byla ohodnocena jedním pracovníkem a odpovědi, které neodpovídaly přiřazeným slovům, byly z datové sady odstraněny.

Podrobnější popis celé datové sady je v [28], odkaz pro stažení datové sady je [29]

Tabulka 5.1: Celkový přehled slov použité datové sady

slovo	počet nahrávek	slovo	počet nahrávek
Backward	1 664	No	3 941
Bed	2 014	Off	3 745
Bird	2 064	On	3 845
Cat	2 031	One	3 890
Dog	2 128	Right	3 778
Down	3 917	Seven	3 998
Eight	3 787	Shiela	2 022
Five	4052	Six	3 860
Folow	1579	Stop	3 872
Forward	1 557	Three	3 727
Four	3 728	Tree	1 759
Go	3 880	Two	3 880
Happy	2 054	Up	3 723
House	2 113	Visual	1 592
Learn	1 575	Wow	2 123
Left	3 801	Yes	4 044
Marvin	2 100	Zero	4 052
Nine	3 934		

Pro trénování navrženého systému bylo vybráno těchto 15 slov: 'Down', 'Five', 'Four', 'Go', 'Left', 'No', 'Off', 'On', 'Right', 'Seven', 'Stop', 'Two', 'Up', 'Yes', 'Zero'.

6 Implementace

Program je logicky rozdělen na dvě části. První část je určena pro vytváření a trénovaní modelu neuronových sítí a druhá pro samotné použití tohoto modelu (rozpoznávání).

Struktura programu je rozdělena do těchto souborů:

- `train.py` – soubor pro spuštění trénování modelu a vytisknutí výsledků trénování
- `nn.py` – v tomto souboru je definováno jak vypadá architektura sítě, vytváří se zde samotný model CNN
- `load.py` – funkce pro načítání dat
- `preprocess.py` – obsahuje funkce pro předzpracování dat a extrakci příznaků
- `save.py` – funkce pro ukládání předzpracovaných dat
- `recognize.py, recoginze_plot.py` – soubory obsahující skripty pro využití již natrénovaného modelu

6.1 Načítání dat

Data jsou načtena z adresáře, který se nachází v proměnné path. Struktura tohoto adresáře musí být vždy taková, že obsahuje podadresáře s názvy rozpoznávaných slov a v těchto podadresářích musí být nahrávky jednotlivých slov. Funkce pro načítání dat jsou v souboru `load.py`.

Načítání je rozděleno do čtyř funkcí. První funkce `'load_dirs'` (výpis 6.1) na základě jednoho vstupního parametru, ve kterém je cesta k datům, načte názvy adresářů (slov) a ty uloží do proměnné, kterou vrátí jako svou návratovou hodnotu.

```
def load_dirs(p):
    try:
        dirs = []
        dirs = [f for f in listdir(p) if (path.isdir(p + '/' + f))]
    except FileNotFoundError:
        print('Nenalezen adresar nebo soubor ' + p)
        sys.exit()
    except:
        raise
        sys.exit()

    return sorted(dirs)
```

Výpis 6.1: Načtení názvů slov

Další funkce `'load_wav'` (výpis 6.2) načítá wav soubory do proměnné, vrací všechna slova v poli a vzorkovací frekvenci. Pokud je načtené slovo kratší než jedna sekunda je vyplněno nulami právě do délky jedné sekundy.

```
def load_wav(p, w):
    try:
        mdata = []
        for filename in glob.iglob(p + '/' + w + '/*.wav'):
            fs, data = wavfile.read(filename)
            if len(data) == 0:
                raise ErrorLenZero
            elif len(data) < 16000:
                data = numpy.pad(data, (0, (16000 - len(data))), 'constant',
                    constant_values=0)
            elif len(data) > 16000:
                    raise ErrorLen
            else:
                data = data.astype(numpy.float64)
                mdata.append(data)
    except ErrorLen:
        print('Nespravny format dat - zvukovy soubor je delsi nez 1s')
        sys.exit()
    except ErrorLenZero:
        print('Nepovedlo se nacist zvukovy soubor')
        sys.exit()
    except:
        raise
        sys.exit()

    return mdata,fs
```

Výpis 6.2: Načtení zvukových souborů

Funkce `'load_labels'` z výpisu 6.3 načítá názvy slov ze souboru `numpy` polí, toto je možné pokud byly slova dříve načteny uloženy do tohoto souboru.

```
def load_labels(path):
    try:
        lbl = []
```

```
        lbl = numpy.load(os.path.dirname(os.path.abspath(inspect.getfile(
            inspect.currentframe()))) + '/labels.npy')
        print('labels loaded!')

    except FileNotFoundError:
        print('Nenalezen soubor se seznamem slov ' + path + '/labels.npy')
        sys.exit()
    except:
        raise
        sys.exit()

    return lbl
```

Výpis 6.3: Načtení názvů slov ze souboru

Předposlední funkce `'load_data'` (výpis 6.4) načítá již předzpracovaná data ze souborů, tyto soubory musí být vždy umístěny ve složkách společně s wav soubory. Tuto funkci lze použít v případě opakovaného spouštění modelu pro stejná data. Načítání a předzpracování je výpočetně a časově náročné a pro účely testování je dobré mít již data předzpracované a šetřit tak čas při ladění modelu. Funkce má na vstupu proměnnou path s cestou k souborům a vrací čtyři proměnné. Funkce vrací pole předzpracovaných dat, dále vrací kategorie (slova), ke kterým data patří, počet rozpoznávaných slov a názvy jednotlivých slov.

```
def load_data(path):
    try:
        lbl = load_labels(path)
        for i, folder in enumerate(lbl):
            mfcctmp = numpy.load(path + '/' + folder + '/' + folder + '.npy')

            if i == 0:
                a = numpy.copy(mfcctmp)
                print(a.shape)
                lbl_i = numpy.zeros(mfcctmp.shape[0])
            else:
                a = numpy.vstack((a, numpy.copy(mfcctmp)))
                print(a.shape)
                lbl_i = numpy.apendix(lbl_i, numpy.full((mfcctmp.shape[0]),
                    fill_value=i))

        a = a.reshape(a.shape[0], 98, 24, 1)
```

```
        lbl_hot = to_categorical(lbl_i)

    except UnboundLocalError:
        print('Nepovedlo se nacist data')
        sys.exit()
    except:
        raise
        sys.exit()

    return a, lbl_hot, i, lbl
```

Výpis 6.4: Načtení předzpracovaných dat ze souboru

Poslední funkce `'get_data'` (výpis 6.5) volá postupně funkce `'load_dirs'` a `'load_wav'` pomocí, kterých načítá zvukové soubory. Dále pro každé slovo zavolá předzpracování, které je popsáno v následující kapitole. Funkce má stejné vstupní parametry i návratové hodnoty jako předchozí funkce `'load_data'`.

```
def get_data(path):
    lbl = load_dirs(path)
    data = []

    try:
        for i, folder in enumerate(lbl):
            data = []
            mfccs = []
            data, fs = load_wav(path, folder)

            for wav_file in data:
                mfccs.append(run_preprocess(fs, wav_file))
            mfcctmp = numpy.asarray(mfccs)
            if i == 0:
                a = numpy.copy(mfcctmp)
                print(a.shape)
                lbl_i = numpy.zeros(mfcctmp.shape[0])
            else:
                a = numpy.vstack((a, numpy.copy(mfcctmp)))
                print(a.shape)
                lbl_i = numpy.append(lbl_i, numpy.full((mfcctmp.shape[0]),
                    fill_value=i))
```

```
        a = a.reshape(a.shape[0], 98, 24, 1)
        lbl_hot = to_categorical(lbl_i)

    except UnboundLocalError:
        print('Nepovedlo se nacist data')
        sys.exit()
    except:
        raise
        sys.exit()

    return a, lbl_hot, i, lbl
```

Výpis 6.5: Načítaní dat

6.2 Předzpracování

Předzpracování je implementováno v souboru `preprocess.py`. Je zde několik funkcí, které jsou postupně volány. Na vstupu těchto funkcí jsou načtená data a jejich vzorkovací frekvence, výstupem jsou pak pole příznaků, které jsou připraveny pro trénování neuronové sítě.

První funkce s názvem `'detrend_data'` (výpis 6.6) má na vstupu načtená data a pouze odstraňuje stejnosměrnou složku signálu. Využívá přitom funkce z knihovny `scipy`. Funkce vrací data bez SS složky.

```
def detrend_data(data):
   data = sig.detrend(data)
   return (data)
```

Výpis 6.6: Funkce pro odstranění stejnosměrné složky

Druhá funkce `'pre_emphasis'` (výpis 6.7) vyrovnává frekvenční charakteristiku řeči jednoduchým filtrem z knihovny `scipy.signal`. Funkce vrací filtrovaná data.

```
def pre_emphasis(data):
   return sig.lfilter(numpy.array([1., -0.97], data.dtype), numpy.array([1.],
      data.dtype), data)
```

Výpis 6.7: Preemfáze

Další funkce `'segmentation'` (výpis 6.8) rozděluje řeč na jednotlivé rámce, vstupními parametry jsou velikost rámce, posun a vstupní data. Výstupem je pak pole polí, ve kterém jsou

uloženy jednotlivé rámce.

```
def segmentation(step, windowsize, data):
   a = []
   for i in range(0, len(data), step):
      if i+windowsize <= len(data):
         a.append(data[i:i+windowsize])
   a = numpy.asarray(a)
   return a
```

Výpis 6.8: Segmentace

Následující dvě funkce (výpis 6.9) mají za úkol aplikovat na jednotlivé rámce z výstupu předchozí funkce Hammingovo okno, které potlačuje okraje a zvýrazňuje střed rámce. První funkce má na vstupu vzorkovací frekvenci a jeden rámec signálu, na který aplikuje Hammingovo okno a výsledek pak vrátí jako návratovou hodnotu. Druhá funkce pak volá tu první pro všechny rámce slova.

```
def hamm(fs, data):
   window = sig.hamming(fs)
   data = numpy.multiply(data,window)
   return data


def hamm_all(fs, data):
   for i in range(0, len(data), 1):
      data[i] = hamm(fs, data[i])
   return data
```

Výpis 6.9: Váhovací funkce

Další funkce počítá rychlou Fourierovu transformaci (výpis 6.10). Vstupními argumenty jsou pole s daty a délka FFT, která musí být vždy mocninou dvojky (2^n). Funkce vrací výkonové spektrum signálu.

```
def fft(data, NFFT):
   mag_f = numpy.absolute(numpy.fft.rfft(data, NFFT))
   pow_f = ((1.0 / NFFT) * ((mag_f) ** 2))
   return pow_f
```

Výpis 6.10: FFT

Následující dvě funkce z výpisu 6.11 mají za úkol samotnou extrakci příznaků. První z funkcí `'filter_banks'` aplikuje banku filtrů na spektrum signálu. Vstupem této funkce je počet filtrů, pole se spektrem jednotlivých rámců řeči, vzorkovací frekvence a délka, na které se prováděla FFT. Výstupem je signál, na kterém je aplikována banka filtrů. Další funkce `'mfcc'` provádí diskrétní kosinovou transformaci, vstupem této funkce je výstup z předchozí funkce. Funkce vrací pole statických Melovských kepstrálních koeficientů pro jednotlivé rámce.

```
def filter_banks(n_filt, pow_f, sample_rate, NFFT):
   low_freq_mel = 0
   high_freq_mel = (2595 * numpy.log10(1 + (sample_rate / 2) / 700))
   mel_points = numpy.linspace(low_freq_mel, high_freq_mel, n_filt + 2)
   hz_points = (700 * (10**(mel_points / 2595) - 1))
   b = numpy.floor((NFFT + 1) * hz_points / sample_rate)

   fbank = numpy.zeros((n_filt, int(numpy.floor(NFFT / 2 + 1))))
   for m in range(1, n_filt + 1):
      f_m_minus = int(b[m - 1]) # left
      f_m = int(b[m])           # center
      f_m_plus = int(b[m + 1]) # right

      for k~in range(f_m_minus, f_m):
         fbank[m - 1, k] = (k~- b[m - 1]) / (b[m] - b[m - 1])
      for k~in range(f_m, f_m_plus):
         fbank[m - 1, k] = (b[m + 1] - k) / (b[m + 1] - b[m])

   filter_banks = numpy.dot(pow_f, fbank.T)
   filter_banks = numpy.where(filter_banks == 0, numpy.finfo(float).eps,
       filter_banks)
   filter_banks = 20 * numpy.log10(filter_banks) # dB

   return filter_banks


def mfcc(fb):
   mfcc = fftp.dct(fb, type=2, axis=1, norm='ortho')[:, 1: 13]
   return mfcc
```

Výpis 6.11: MFCC

Posledním krokem předzpracování je výpočet dynamických koeficientů. Funkce (výpis 6.12) má jeden vstupní parametr a tím je pole příznaků pro zpracovávané slovo. Funkce pak vypočte dynamické koeficienty z 5 rámců (2 rámce před aktuálním, 2 rámce za aktuálním + aktuálně zpracovávaný rámec) a na výstupu vrací pole, které kromě statických příznaků obsahuje i ty dynamické.

```
def delta(feat):
   N = 2
   NUMFRAMES = len(feat)
   denominator = 2 * sum([i**2 for i in range(1, N+1)])
   delta_f = numpy.empty_like(feat)
   padded = numpy.pad(feat, ((N, N), (0, 0)), mode='edge') # padded version of
       feat
   for t in range(NUMFRAMES):
      delta_f[t] = numpy.dot(numpy.arange(-N, N+1), padded[t: t+2*N+1]) /
          denominator # [t : t+2*N+1] == [(N+t)-N : (N+t)+N+1]
   out_feat = []
   for i in range(NUMFRAMES):
      out_feat.append(numpy.concatenate((feat[i], delta_f[i])))
   out_feat = numpy.asarray(out_feat)
   return out_feat
```

Výpis 6.12: Výpočet dynamických koeficientů

Poslední funkce `'run_preprocess'` (výpis 6.13) má na vstupu načtený wav soubor a postupně zavolá všechny funkce a vrátí vypočtené příznaky.

```
def run_preprocess(fs, data):
    data = detrendData(data)
    data = pre_emphasis(data)
    data = segmentation(fs // 100, 400, data)
    data = hamm_all(400, data)
    pow_spc = fft(data, 1024)
    flt = filter_banks(26, pow_spc, fs, 1024)
    mfccs = mfcc(flt)
    dltc = delta(mfccs)
    return dltc
```

Výpis 6.13: Spuštění předzpracování

Pro konkrétně použitou datovou sadu, která obsahuje wav soubory o délce 1 s se vzorkovací frekvencí 16 KHz byla použita délka okna 25 ms s 10 ms posunem. Každé slovo je takto rozděleno na 98 rámců a pro každý rámec je vypočteno 12 statických příznaků a 12 dynamických příznaků.

6.3 Ukládání dat

Ukládání je realizováno v souboru `save.py`. Tento soubor obsahuje dvě funkce, z nichž první je `'save_data'` (výpis 6.14). Tato funkce má na vstupu cestu k datům a má za úkol data načíst, provést předzpracování a uložit pole příznaků pro souborů. Pole příznaků jsou vždy uložena do adresáře, kde se nachází samotné zvukové soubory.

```
def save_data(path):
    dirs = []
    dirs = l.load_dirs(path)
    data = []

    for i, folder in enumerate(dirs):
        data = []
        mfccs = []
        data = l.load_wav(path, folder)
        for wav_file in data:
            mfccs.append(run_preprocess(fs, wav_file))
        mfcctmp = numpy.asarray(mfccs)
        numpy.save(path + '/' + folder + '/' + folder + '.npy', mfcctmp)
        print(folder + ' saved!')
```

Výpis 6.14: Načtení, předzpracování a uložení příznaků do souboru

Druhá funkce (výpis 6.15) slouží pro uložení seznamu slov, funkce pouze načte seznam adresářů a ten uloží do souboru.

```
def save_labels(path):
    lbl = []
    lbl = l.load_dirs(path)

    numpy.save(os.path.dirname(os.path.abspath(inspect.getfile(inspect.
        currentframe()))) + '/labels.npy', numpy.asarray(lbl))
    print('labels saved!')
```

Výpis 6.15: Uložená seznamu slov do souboru

6.4 Model CNN

Samotný model neuronové sítě je uložen v souboru `'nn.py'`. Pro vytváření a trénování modelu bylo použito aplikační rozhraní Keras (podrobnější popis v kapitole 5.1). Byl použit sekvenční model s těmito vrstvami:

- Dense – toto znamená v Kerasu plně propojenou vrstvu
- Dropout – vrstva pro předcházení přeučení sítě
- Flatten – vrstva, která má za úkol vyrovnávat vstup pro další vrstvy
- Conv2D – konvoluční vrstva
- MaxPooling2D – pooling vrstva
- Activation – aktivační vrstva

Účel jednotlivých vrstev je vysvětlen v kapitole 4.3.1.

Výpis 6.16 obsahuje kód pro vytvoření modelu. Poslední řádek funkce kompiluje model a připravuje ho tak pro trénování. Na obrázku 6.1 je naznačena jeho architektura.

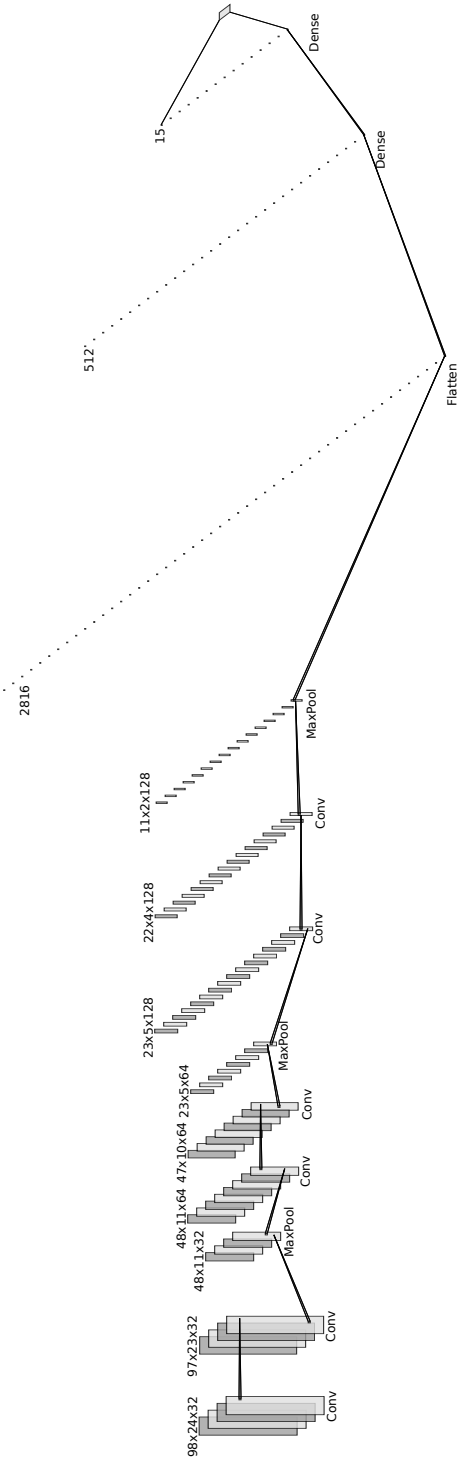
```
def getmodel(dense):
    model = Sequential()
    model.add(Conv2D(32, (2, 2), padding='same', input_shape=(98, 24, 1)))
    model.add(Activation('relu'))
    model.add(Dropout(0.1))
    model.add(Conv2D(32, (2, 2)))
    model.add(Activation('relu'))
    model.add(Dropout(0.1))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(GaussianNoise(0.1))
    model.add(Activation('relu'))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (2, 2), padding='same'))
    model.add(Activation('relu'))
    model.add(Dropout(0.1))
    model.add(Conv2D(64, (2, 2)))
    model.add(Activation('relu'))
    model.add(Dropout(0.1))
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
    model.add(GaussianNoise(0.1))
    model.add(Activation('relu'))
    model.add(Dropout(0.25))

    model.add(Conv2D(128, (2, 2), padding='same'))
    model.add(Activation('relu'))
    model.add(Dropout(0.1))
    model.add(Conv2D(128, (2, 2)))
    model.add(Activation('relu'))
    model.add(Dropout(0.1))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(GaussianNoise(0.1))
    model.add(Activation('relu'))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(dense))
    model.add(Activation('softmax'))

    model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.
        optimizers.Adam(), metrics=['accuracy'])

    return model
```

Výpis 6.16: Model neuronové sítě

Vytvořený model začíná konvoluční vrstvou, kterou následují aktivační a dropout vrstvy – toto je zde dvakrát za sebou. Poté je zde pooling vrstva, po které je přidán šum pro zvýšení robustnosti rozpoznávání a následuje opět aktivační a dropout vrstva. Celá tato struktura se třikrát opakuje.

Na konci modelu je pak vyrovnávací vrstva, plně propojená vrstva, aktivační a dropout vrstvy, poslední je pak plně propojená vrstva, která má stejný počet neuronů jako je počet rozpoznávaných tříd (slov), za touto vrstvou pak následuje ještě jedna aktivační vrstva.



Obrázek 6.1: Architektura použité CNN

6.5 Trénování

Trénování lze spustit v souboru `train.py`. V tomto souboru je definována cesta k datům v proměnné path. Před trénováním je potřeba načíst data a ty rozdělit do tří množin.

Celý program je možno spustit souborem `train.py`, který vyžaduje několik parametrů. První sada parametrů je nepovinná a lze použít jeden z těchto tří:

- parametr -l (--load) – slouží pro načtení již vytvořených příznaků pro trénování modelu
- parametr -s (--save) – načte data, vypočte z nich příznaky a ty uloží do souborů, poté se spustí trénování modelu
- parametr -h (--help) – vypisuje nápovědu

Pokud není zadán ani jeden z těchto přepínačů, použije načítání zvukových souborů s předzpracováním a extrakcí příznaků bez jejich uložení. Jediný povinný parametr je cesta k datové sadě, která musí být vždy zadána.

Ve výpisu 6.17 je realizováno čtení vstupních parametrů pro spuštění trénovaní modelu. Na obrázku 6.2 je zobrazen výpis nápovědy.

```
parser = argparse.ArgumentParser()
group = parser.add_mutually_exclusive_group()
group.add_argument('-l', '--load', help='Nacteni priznaku ze souboru', action="
    store_true")
group.add_argument('-s', '--save', help='Nacteni dat, extrakce priznaku a
    ulozeni do souboru', action="store_true")
parser.add_argument("path", help="Cesta k~datove sade")
args = parser.parse_args()

if args.load:
    data, data_lbl, classes_count, labels = load_data(args.path)
    run_train(data, data_lbl, classes_count, labels)

elif args.save:
    save_data(args.path)
    save_labels(args.path)
    data, data_lbl, classes_count, labels = load_data(args.path)
    run_train(data, data_lbl, classes_count, labels)

else:
```

```
    data, data_lbl, classes_count, labels = get_data(args.path)
    save_labels(args.path)
    run_train(data, data_lbl, classes_count, labels)
```

Výpis 6.17: Parser vstupních parametrů

Další parametr je povinný, je to cesta k datové sadě, která musí být vždy zadána. Požadavky na to jak musí být data rozdělena jsou popsány v kapitole 6.1.

```
usage: train.py [-h] [-l | -s] path

positional arguments:
  path        Cesta k datové sadě

optional arguments:
  -h, --help  show this help message and exit
  -l, --load  Načtení příznaků ze souborů
  -s, --save  Načtení dat, extrakce příznaků a uložení do souborů
```

Obrázek 6.2: Výpis nápovědy pro použití trénovacího programu

Poté co se určí jakým způsobem se budou načítat data je zavolána funkce `'run_train'`, vstupními parametry této funkce jsou pole příznaků a k nim třídy, do kterých patří, počet slov a jejich seznam.

Data jsou rozdělena funkcí `'train_test_split()'` z knihovny `sklearn`. Data jsou náhodně rozdělena na trénovací, testovací a validační množinu. Trénovací data obsahují 80% datové sady a testovací a validační množina množina pak mají obě 10% datové sady. Poslední množina není nutná pro fungování modelu, ale je na ní možno ověřit, že model není přeučený.

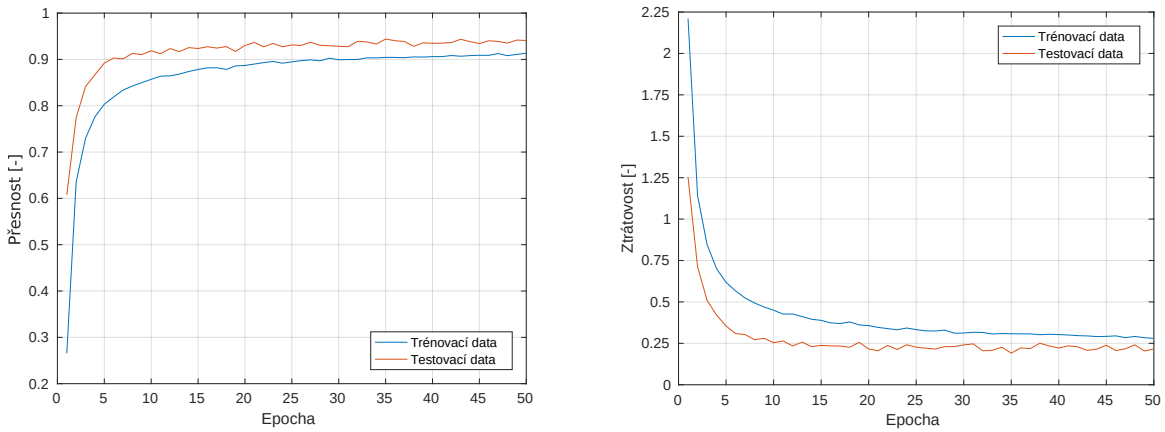
Trénování modelu lze zahájit jednoduše spuštěním příkazu `'model.fit()'` z Kerasu. Tato funkce se volá pro vytvořený model, vstupem je trénovací množina, validační množina, počet epoch, počet slov použitých pro jeden průchod sítí, metrika případně další volitelné parametry. Průběh trénování pro každou epochu je vždy zobrazován v přesnosti rozpoznávání pro trénovací a testovací množinu dat. Funkce `'model.fit()'` vrací objekt typu `History`, který obsahuje informace o průběhu trénování. Díky tomuto objektu je pak možné vytvořit grafy průběhu přesnosti a ztrátovosti během učení.

Výpis 6.18 ukazuje část funkce `'run_train'`, která rozděluje data do tří množin, volá funkci pro vytvoření a zkompilování modelu sítě a spouští trénování. Výsledek je pak uložen v proměnné history a její obsah je vykreslen na obrázku 6.3.

```
train, test, train_lbl, test_lbl = train_test_split(x, x_lbl, test_size=0.2,
    random_state=random.randint(1, 30))
test, val, test_lbl, val_lbl = train_test_split(test, test_lbl, test_size=0.5,
    random_state=random.randint(31, 60))
```

```
model = nn.getmodel(dense + 1)

history = model.fit(train, train_lbl, batch_size=100, epochs=2, verbose=1,
    validation_data=(test, test_lbl))
```

Výpis 6.18: Rozdělení dat a spuštění trénování



(a) Přesnost

(b) Ztrátovost

Obrázek 6.3: Průběh trénování při 50 epochách

Na obrázku 6.3 je vykreslen průběh učení neuronové sítě pro 50 epoch. Bylo zde použito dohromady 46604 slov pro trénování a 5826 slov pro testování. Výsledná přesnost modelu byla 91.35% pro trénovací množinu a 94.08% pro testovací množinu.

Trénování neuronové sítě probíhalo na počítači s procesorem Intel i5-6600k a 16 GB operační paměti na operačním systému Xubuntu 16.04. Doba trénování byla přibližně 3 hodiny.

Pro lepší přehled o výsledcích trénování modelu je možné vytvořit tzv. confusion matrix – matici, která ukazuje výsledky rozpoznávání pro jednotlivá slova. Obrázek 6.4 obsahuje právě takovouto matici na trénovací množině dat. V řádcích jsou zde skutečné počty slov a ve sloupcích je pak vidět kolikrát které slovo model určil správně a kolikrát ho přiřadil do jiné třídy slov. Z toho vyplývá, že na diagonále jsou správně rozpoznaná slova a těch by při dobrém natrénování mělo být nejvíce. V posledním řádku i sloupci jsou součty všech slov.

Predicted	down	five	four	go	left	no	off	on	right	seven	stop	two	up	yes	zero	__all__
Actual																
down	3010	1	1	12	3	29	1	0	1	0	2	2	11	1	0	3074
five	1	3220	2	3	5	10	3	9	5	1	6	0	7	0	1	3273
four	4	10	2881	6	4	14	9	15	2	6	3	9	11	1	4	2979
go	44	0	4	2983	2	48	0	1	2	1	2	18	9	0	0	3114
left	1	2	0	3	2948	19	0	2	4	1	1	5	4	19	0	3009
no	25	0	0	13	5	3110	0	0	0	0	2	5	6	3	2	3171
off	6	7	10	5	9	11	2865	34	0	1	5	3	29	2	1	2988
on	22	15	11	4	4	11	23	2959	0	4	1	5	14	0	1	3074
right	5	33	0	2	15	5	0	1	2919	2	0	3	9	3	0	2997
seven	4	3	2	2	4	0	1	1	2	3181	10	6	6	3	4	3229
stop	6	3	1	0	2	5	3	0	3	11	3065	5	7	5	2	3118
two	4	1	1	8	5	15	0	0	1	2	0	3052	9	0	11	3109
up	12	30	2	32	25	27	295	60	0	11	58	8	2470	2	3	3035
yes	0	0	0	1	5	13	1	1	0	1	1	1	2	3205	1	3232
zero	4	2	0	5	6	24	0	0	0	9	4	24	6	7	3111	3202
__all__	3148	3327	2915	3079	3042	3341	3201	3083	2939	3231	3160	3146	2600	3251	3141	46604

Obrázek 6.4: Confusion matrix pro trénovací data

Vykreslení confusion matrix je ve výpisu 6.19. Pro vytvoření této matice je potřeba mít predikce z modelu, to je možné zavoláním funkce `'model.predict()'` z knihovny Keras. Výsledkem funkce `'model.predict()'` je vždy vektor, který má délku stejnou jako je počet rozpoznávaných slov s pravděpodobnostmi. Z tohoto vektoru je vždy vybrána ta pravděpodobnost s největší hodnotou a podle indexu, na kterém se nachází je jí přiřazeno odpovídající slovo. Vstupem funkce pro vytvoření matice jsou pak původní hodnoty z načtených dat a hodnoty z predikce modelu.

```
pred_train = model.predict(train)
cm_train = ConfusionMatrix(train_lbl.argmax(axis=1), pred_train.argmax(axis=1),
    labels=lbl)
print('Confusion matrix for training data')
print(cm_train)
print('\n\n')
```

Výpis 6.19: Vypsání confusion matrix

Na obrázcích 6.5 a 6.6 jsou pak výsledky rozpoznávání pro testovací a validační množinu. Pro vytvoření matic byla použita knihovna `pandas_ml`, kromě těchto matic lze pak získat i statistiky rozpoznávání pro jednotlivá slova. Tyto statistiky mohou být užitečné při ladění modelu.

Predicted	down	five	four	go	left	no	off	on	right	seven	stop	two	up	yes	zero	__all__
Actual																
down	394	1	0	3	4	3	0	0	0	0	0	1	1	1	0	408
five	1	376	1	0	2	3	0	2	2	1	1	0	2	0	0	391
four	2	2	352	0	3	3	3	5	0	1	1	0	0	0	2	374
go	7	1	1	339	1	9	1	0	0	0	2	3	1	2	0	367
left	0	3	0	0	379	5	0	0	1	1	0	2	2	8	0	401
no	9	0	0	2	3	368	0	0	0	0	0	0	0	0	0	382
off	1	3	2	1	2	4	312	7	0	0	3	0	9	1	0	345
on	4	8	3	1	0	1	2	366	0	0	1	0	2	0	0	388
right	3	6	0	0	4	3	0	0	371	0	0	1	3	1	1	393
seven	2	0	0	0	1	2	0	0	1	383	7	1	1	0	1	399
stop	4	1	1	2	1	1	1	1	0	1	365	1	3	0	1	383
two	0	1	1	3	2	2	0	0	0	4	1	386	1	0	3	404
up	2	6	1	8	0	6	39	8	0	0	5	0	253	0	0	328
yes	0	0	0	1	6	3	0	0	0	0	0	0	2	407	0	419
zero	0	0	0	0	1	1	0	0	0	2	0	8	1	1	430	444
__all__	429	408	362	360	409	414	358	389	375	393	386	403	281	421	438	5826

Overall Statistics:

Accuracy: 0.9407826982492276

Obrázek 6.5: Confusion matrix pro testovací data

Predicted	down	five	four	go	left	no	off	on	right	seven	stop	two	up	yes	zero	__all__
Actual																
down	408	0	1	7	1	5	1	0	0	1	3	2	5	1	0	435
five	3	373	1	0	1	3	0	1	0	1	1	2	1	0	1	388
four	0	0	354	2	1	1	4	4	0	2	1	5	0	1	0	375
go	13	0	2	358	1	18	2	0	1	0	0	2	1	0	1	399
left	0	1	0	0	368	4	0	0	1	0	0	1	5	11	0	391
no	5	1	0	7	2	367	0	0	0	0	2	3	0	0	1	388
off	0	1	0	1	2	4	381	10	1	0	1	2	8	1	0	412
on	6	4	6	3	0	5	2	353	0	1	1	1	1	0	0	383
right	0	12	0	0	4	2	0	0	368	1	0	0	1	0	0	388
seven	3	5	0	0	0	1	1	0	0	348	4	3	1	2	2	370
stop	0	3	1	1	1	0	1	1	0	6	350	2	4	0	1	371
two	0	1	2	2	2	4	1	0	0	3	0	348	2	1	1	367
up	6	5	1	12	5	3	53	14	0	2	6	1	252	0	0	360
yes	0	0	0	0	3	1	0	0	0	1	0	2	1	383	2	393
zero	0	0	0	1	3	6	0	0	1	3	0	2	1	4	385	406
__all__	444	406	368	394	394	424	446	383	372	369	369	376	283	404	394	5826

Overall Statistics:

Accuracy: 0.9261929282526605

Obrázek 6.6: Confusion matrix pro validační data

6.6 Rozpoznávání

Druhá část programu má za úkol využití natrénovaného modelu. Jsou zde dva skripty, které berou jako vstup zvuk z mikrofonu. Jeden z nich pouze vypisuje do konzole rozpoznané slovo a druhý k tomu má ještě grafické okno, kde vykresluje aktuální průběh vstupního signálu a rozpoznané slovo.

Obě aplikace předpokládají, že model a seznam slov je uložen ve stejném adresáři, ze kterého je skript spuštěn.

6.6.1 Konzolová aplikace

Tato část programu využívá knihovnu `pyaudio`, která dokáže snímat zvuk z mikrofonu.

Na začátku programu je nastavení proměnných pro začátek snímání z mikrofonu. Poté zde běží smyčka (výpis 6.20), která dokud je mikrofon aktivní, bere data ze vstupu se vzorkovací frekvencí 16 KHz a po načtení určité části dat provede předzpracování a jeho výsledek předá natrénovanému modelu. Z této predikce je pak vybráno slovo s největší pravděpodobností a to je vypsáno do konzole. Pokud jsou všechny pravděpodobnosti menší než 0.6 tak je slovo klasifikováno jako unknown – nerozpoznáno.

Ve smyčce se provede předzpracování pokud je načteno alespoň 16000 vzorků. Na začátku tedy program počká, dokud není načtena 1 s zvuku a poté začne rozpoznávat. Na konci provedení každé smyčky je vždy umazána část pole s daty, na kterých se provádí předzpracování, tato velikost je v proměnné buff a je nastavena na 600 vzorků. Program vždy smaže dvojnásobek velikosti této proměnné, počká než bude mít opět načteno 16000 vzorků a opět zahájí rozpoznávání. Reálně to tedy znamená že program každých 75 ms hledá, jestli se ve vstupním signálu nachází slovo.

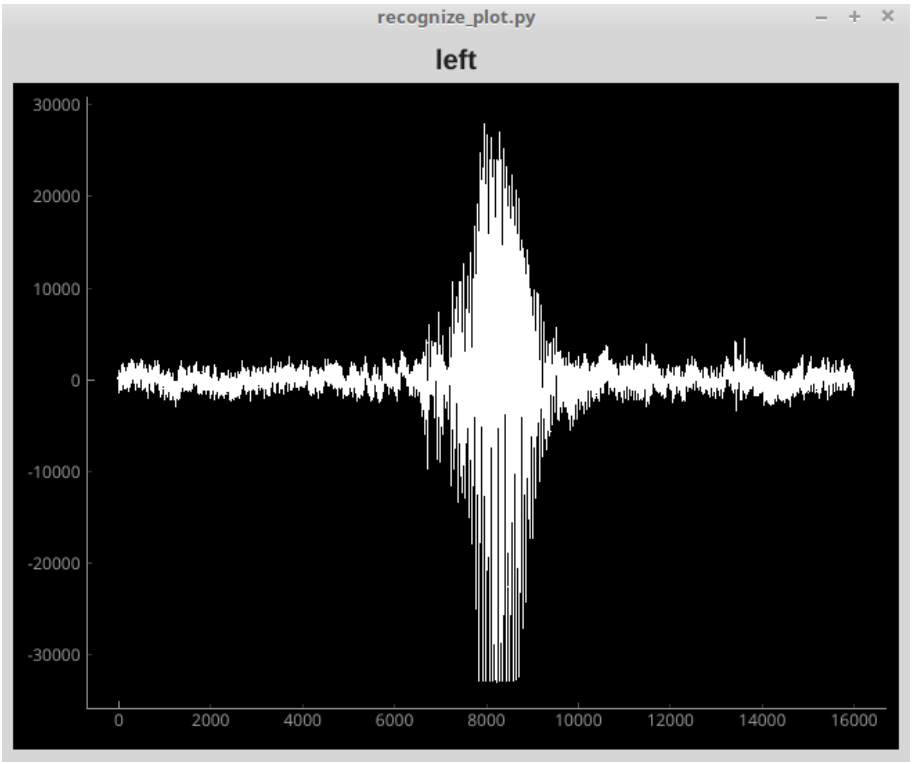
```
while stream.is_active():
    item = q.get()
    if item is None:
        break
    data = numpy.append(data, item)

    if len(data) > 15999:
        mfcc = run_preprocess(fs, data[0:16000])
        mfcc = mfcc.reshape(1, 98, 24, 1)
        prediction = model.predict(mfcc)
        out = numpy.argmax(prediction)
        if prediction[0][out] > 0.60:
            sys.stdout.write('\033[K')
            print('best prediction is: ' + lbl[:][out], end='\r')
        else:
            sys.stdout.write("\033[K")
            print('best prediction is: unknown', end='\r')
        data = numpy.delete(data, numpy.s_[0:(2*buff)])

    q.task_done()
```

Výpis 6.20: Rozpoznávání z mikrofonu

6.6.2 Grafická aplikace

Grafická aplikace používá pro rozpoznávání stejný princip jako ta konzolová z předchozí kapitoly, ale navíc je zde knihovnou `pyqtgraph` vytvořeno grafické okno, které vykresluje aktuální průběh vstupního signálu. Okno je vytvořeno pomocí třídy, ve které jsou definovány jednotlivé prvky i funkce pro vykreslování.

Vzhled grafické aplikace je na obrázku 6.7. Skládá se ze řádku, na kterém je vypsáno rozpoznané slovo a pod ním je vždy aktuální průběh signálu z mikrofonu.



Obrázek 6.7: Vzhled grafické aplikace pro rozpoznávání

Závěr

V této práci jsem se věnovala využití metod strojového učení pro automatické rozpoznávání řeči. Cílem bylo navrhnout a vytvořit systém pro automatické rozpoznávání izolovaných slov.

Součástí práce je rešerše, kde jsou popsány již existující systémy ASR. Na základě rešerše byly vybrány metody pro realizaci vlastního programu. Důležitou součástí práce je také teoretická část, která seznamuje čitatele s metodami, které se používají pro zpracovávání řeči. Patří zde předzpracování, které připravuje řeč do podoby vhodné pro zpracovávání počítačem. Jsou zde popsány dvě metody pro extrakci příznaků a to lineární prediktivní kódování (LPC) a melovské kepstrální koeficienty (MFCC). Poslední část teorie se zabývá obecně strojovým učením a metodami, které se používají v systémech automatického rozpoznávání řeči. Jsou zde popsány algoritmy dynamického borcení času, skryté Markovovy modely a konvoluční neuronové sítě.

Praktická část byla věnována realizaci ASR systému pro izolovaná slova. Byly použity nástroje Keras a TensorFlow pro programovací jazyk Python. Pro trénování a testování celého systému byla využita databáze obsahující dohromady 105 829 nahrávek 35 slov od 2618 různých řečníků (žen i mužů) v různých prostředích. Pro vytvářený systém bylo vybráno 15 slov. Databáze byla náhodně rozdělena na trénovací, testovací a validační množinu. Trénovací data obsahuji 80% nahrávek. Zbylých 20% je rozděleno do testovací a validační množiny stejným poměrem.

Předzpracování signálu začíná odstraněním stejnosměrné složky a preemfází. Dalším krokem je segmentace, kde bylo každé slovo rozděleno na úseky dlouhé 25 ms s překryvem 10 ms. Délka každé nahrávky je 1 s, takže výsledkem bylo 98 rámců pro každé slovo. Posledním krokem předzpracování bylo váhování Hammingovým oknem. Pro extrakci příznaků bylo použito MFCC s delta koeficienty. Z každého rámce se extrahuje 12 MFCC příznaků a 12 delta koeficientů, to znamená, že pro každé slovo je vytvořena matice 98x24 příznaků.

Následně byly použity různé modely za účelem dosažení co nejvyšší přesnosti rozpoznávání. Finální model neuronové sítě (obrázek 6.1) obsahuje 6 konvolučních vrstev, 11 aktivačních vrstev, 10 dropout vrstev, 3 pooling vrstvy, 3 vrstvy pro přidání šumu, jednu vyrovnávací vrstvu a 2 plně propojené vrstvy. Jeho natrénování zabralo přibližně 3h na počítači s procesorem Intel i5-6600k a 16 GB operační paměti na operačním systému Xubuntu 16.04. Popsaný model dosáhl přesnosti 91.35% pro trénovací množinu a 94.08% pro testovací množinu na datové sadě obsahující 15 slov.

Natrénovaný systém je schopen rozpoznávání klíčových slov z řeči pomocí mikrofonu v reálném čase. Rozpoznávání je možné pomocí konzolové aplikace nebo grafické aplikace, která zároveň vykresluje časový průběh zvuku snímaného z mikrofonu.

Vytvořený model a jeho efektivita potvrzuje popularitu hlubokých neuronových sítí v různých odvětvích. Systém je navržen tak, aby byl snadno rozšiřitelný pro další použití.

Systém ASR popsaný v této práci může být nasazen v různých oblastech Industry 4.0, kde přináší výhody využití pro automatizaci, má uživatelsky přívětivé rozhraní a je snadno použitelný.

Literatura

[1] IMTIAZ, Muhammad Atif a Gulistan RAJA. Isolated word Automatic Speech Recognition (ASR) System using MFCC, DTW & KNN.In: *2016 Asia Pacific Conference on Multimedia and Broadcasting (APMediaCast)* [online]. IEEE, 2016, 2016, s. 106-110 [cit. 2018-03-13]. DOI: 10.1109/APMediaCast.2016.7878163. ISBN 978-1-4673-9791-9. Dostupné z: `http://ieeexplore.ieee.org/document/7878163/`

[2] SENTHILDEVI K. A a CHANDRA E. Keyword spotting system for Tamil isolated words using Multidimensional MFCC and DTW algorithm. In: *2015 International Conference on Communications and Signal Processing (ICCSP)* [online]. IEEE, 2015, 2015, s. 0550-0554 [cit. 2019-03-25]. DOI: 10.1109/ICCSP.2015.7322545. ISBN 978-1-4799-8081-9. Dostupné z: `http://ieeexplore.ieee.org/document/7322545/`

[3] XU, Lijun a Minyi KE. Research on isolated word recognition with DTW-based. In: *2012 7th International Conference on Computer Science & Education (ICCSE)* [online]. IEEE, 2012, 2012, s. 139-141 [cit. 2019-03-25]. DOI: 10.1109/ICCSE.2012.6295044. ISBN 978-1-4673-0242-5. Dostupné z: `http://ieeexplore.ieee.org/document/6295044/`

[4] ABU SHARIAH, Mohammad A. M., Raja N. AINON, Roziati ZAINUDDIN a Othman O. KHALIFA. Human computer interaction using isolated-words speech recognition technology. In: *2007 International Conference on Intelligent and Advanced Systems* [online]. IEEE, 2007, 2007, s. 1173-1178 [cit. 2018-03-13]. DOI: 10.1109/ICIAS.2007.4658569. ISBN 978-1-4244-1355-3. Dostupné z: `http://ieeexplore.ieee.org/document/4658569/`

[5] DHAVALE DHANASHRI a S. B. DHONDE. Isolated Word Speech Recognition System Using Deep Neural Networks. SATAPATHY, Suresh Chandra, Vikrant BHATEJA a Amit JOSHI, ed. *Proceedings of the International Conference on Data Engineering and Communication Technology* [online]. Singapore: Springer Singapore, 2017, 2017-08-24, s. 9-17 [cit. 2018-06-14]. Advances in Intelligent Systems and Computing. DOI: 10.1007/978-981-10-1675-2_2. ISBN 978-981-10-1674-5. Dostupné z: `http://link.springer.com/10.1007/978-981-10-1675-2_2`

[6] RANJAN, Rajeev a Rajesh Kumar DUBEY. Isolated Word Recognition using HMM for Maithili dialect. In: *2016 International Conference on Signal Processing and Communication (ICSC)* [online]. IEEE, 2016, 2016, s. 323-327 [cit. 2019-03-25]. DOI: 10.1109/ICSPCom.2016.7980600. ISBN 978-1-5090-2684-5. Dostupné z: `http://ieeexplore.ieee.org/document/7980600/`

[7] FRANGOULIS, E. Isolated word recognition in noisy environment by vector quantization of the HMM and noise distributions. In: *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing* [online]. IEEE, 1991, 1991, 413-416

vol.1 [cit. 2019-03-25]. DOI: 10.1109/ICASSP.1991.150364. ISBN 0-7803-0003-3. Dostupné z: http://ieeexplore.ieee.org/document/150364/

[8] ZHAO, Lishuang a Zhiyan HAN. Speech Recognition System Based on Integrating Feature and HMM. In: *2010 International Conference on Measuring Technology and Mechatronics Automation* [online]. IEEE, 2010, 2010, s. 449-452 [cit. 2019-03-25]. DOI: 10.1109/ICMTMA.2010.298. ISBN 978-1-4244-5001-5. Dostupné z: http://ieeexplore.ieee.org/document/5458876/

[9] SINGHAL, Shweta a Rajesh Kumar DUBEY. Automatic speech recognition for connected words using DTW/HMM for English/ Hindi languages. In: *2015 Communication, Control and Intelligent Systems (CCIS)* [online]. IEEE, 2015, 2015, s. 199-203 [cit. 2018-03-13]. DOI: 10.1109/CCIntelS.2015.7437908. ISBN 978-1-4673-7541-2. Dostupné z: http://ieeexplore.ieee.org/document/7437908/

[10] T. Sainath and C. Parada. Convolutional neural networks for small-footprint keyword spotting. *Proc. Interspeech*, 2015.

[11] G. Chen, C. Parada and G. Heigold, Small-footprint keyword spotting using deep neural networks, *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Florence, 2014, pp. 4087-4091. doi: 10.1109/ICASSP.2014.6854370 [cit. 2019-03-09]. Dostupné z http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6854370&isnumber=6853544

[12] UHLÍŘ, Jan. *Technologie hlasových komunikací.* Praha: Nakladatelství ČVUT, 2007. ISBN 978-80-01-03888-8

[13] PSUTKA, Josef. *Mluvíme s počítačem česky.* Praha: Academia, 2006. ISBN 80-200-1309-1

[14] PARTILA, Pavol, Miroslav VOZNAK, Martin MIKULEC a Jaroslav ZDRALEK. Fundamental Frequency Extraction Method using Central Clipping and its Importance for the Classification of Emotional State. *Advances in Electrical and Electronic Engineering* [online]. 2012, 10(4), 270 - 275 [cit. 2019-04-04]. DOI: 10.15598/aeee.v10i4.738. ISSN 1804-3119. Dostupné z: http://advances.utc.sk/index.php/AEEE/article/view/738

[15] Zaccone, G., Karim, M. and Menshawy, A. *Deep learning with TensorFlow.* ISBN-978-1-78646-978-6

[16] SAKOE, H. a S. CHIBA. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* [online]. 1978, 26(1), 43-49 [cit. 2018-11-15]. DOI: 10.1109/TASSP.1978.1163055. ISSN 0096-3518. Dostupné z: http://ieeexplore.ieee.org/document/1163055/

[17] RABINER, L.R. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* [online]. 77(2), 257-286 [cit. 2018-11-05]. DOI: 10.1109/5.18626. ISSN 00189219. Dostupné z: `http://ieeexplore.ieee.org/document/18626/`

[18] YOUNG, Steve, Gunnar EVERMANN, Mark GALES, Thomas HAIN, Dan KERSHAW, Xunying (Andrew) LIU, Gareth MOORE, Julian ODELL, Dave OLLASON, Dan POVEY, Valtcho VALTCHEV a Phil WOODLAND, 2006. *The HTK Book*. HTK Version 3.4. B.m.: Cambridge University Engineering Department.

[19] DORFLER, Monika, Roswitha BAMMER a Thomas GRILL. Inside the spectrogram: Convolutional Neural Networks in audio processing. In: *2017 International Conference on Sampling Theory and Applications (SampTA)* [online]. IEEE, 2017, 2017, s. 152-155 [cit. 2019-04-04]. DOI: 10.1109/SAMPTA.2017.8024472. ISBN 978-1-5386-1565-2. Dostupné z: `http://ieeexplore.ieee.org/document/8024472/`

[20] GOUDA, Sanjay Krishna, et al. Speech Recognition: Keyword Spotting Through Image Recognition. *arXiv preprint arXiv:1803.03759*, 2018.

[21] FU, Szu-Wei, et al. Complex spectrogram enhancement by convolutional neural network with multi-metrics learning. In: *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2017. p. 1-6.

[22] ABDEL-HAMID, Ossama, Abdel-rahman MOHAMED, Hui JIANG, Li DENG, Gerald PENN a Dong YU. Convolutional Neural Networks for Speech Recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* [online]. 2014, 22(10), 1533-1545 [cit. 2019-03-25]. DOI: 10.1109/TASLP.2014.2339736. ISSN 2329-9290. Dostupné z: `http://ieeexplore.ieee.org/document/6857341/`

[23] PARK, Sunchan, Yongwon JEONG a Hyung Soon KIM. Multiresolution CNN for reverberant speech recognition. In: *2017 20th Conference of the Oriental Chapter of the International Coordinating Committee on Speech Databases and Speech I/O Systems and Assessment (O-COCOSDA)* [online]. IEEE, 2017, 2017, s. 1-4 [cit. 2019-03-25]. DOI: 10.1109/ICSDA.2017.8384470. ISBN 978-1-5386-3333-5. Dostupné z: https://ieeexplore.ieee.org/document/8384470/

[24] Srivastava Nitish, Hinton Geoffrey, Krizhevsky Alex, Sutskever Ilya, Salakhutdinov Ruslan. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research.* 15. 1929-1958.

[25] HUANG, Jui-Ting, Jinyu LI a Yifan GONG. An analysis of convolutional neural networks for speech recognition. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* [online]. IEEE, 2015, 2015, s. 4989-4993 [cit.

2019-03-25]. DOI: 10.1109/ICASSP.2015.7178920. ISBN 978-1-4673-6997-8. Dostupné z: http://ieeexplore.ieee.org/document/7178920/

[26] LECUN, Y., L. BOTTOU, Y. BENGIO a P. HAFFNER. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* [online]. 86(11), 2278-2324 [cit. 2019-03-09]. DOI: 10.1109/5.726791. ISSN 00189219. Dostupné z: http://ieeexplore.ieee.org/document/726791/

[27] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software dostupný z https://tensorflow.org.

[28] Pete Warden. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. [online] [cit. 2019-03-09]. Dostupné z http://arxiv.org/abs/1804.03209

[29] (2018) Speech commands dataset version 2. [online] [cit. 2019-03-25]. Dostupné z: http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz

A Příloha v IS EDISON

Tato příloha obsahuje soubory se zdrojovými kódy, natrénovaný model, soubor se seznamem slov potřebný pro rozpoznávání a dokumentaci.

Seznam souborů:

- zdrojové kódy – load.py, nn.py, preprocess.py, recognize.py, recognize_plot.py, save.py, train.py
- model a seznam slov – model.h5, labels.npy
- dokumentace – dokumentace.pdf

B Programová a uživatelská dokumentace

Rozpoznávání izolovaných slov dokumentace

Vydání 0.1

Martina Slívová

18.04.2019

Obsah

1 Popis programu **2**
1.1 Použité nástroje . . . 2

2 Programová dokumentace **3**
2.1 Načítání dat . . . 3
2.2 Předzpracování a extrakce příznaků . . . 4
2.3 Ukládání dat . . . 5
2.4 Vytvoření modelu neuronové sítě . . . 6
2.5 Trénování modelu neuronové sítě . . . 6
2.6 Grafická aplikace pro rozpoznávání . . . 7
2.7 Konzolová aplikace pro rozpoznávání . . . 7

3 Uživatelská dokumentace **9**
3.1 Příprava modelu - trénování . . . 9
3.2 Rozpoznávání . . . 9

4 Indexy a tabulky **10**

Rejstřík modulů Pythonu **11**

Rejstřík **12**

Toto je dokumentace k programu, který byl vytvořen v rámci diplomové práce Využití metod strojového učení pro rozpoznávání řeči.

KAPITOLA 1

Popis programu

Program je logicky rozdělen na dvě části. První část je určena pro vytváření a trénovaní modelu neuronových sítí a druhá pro samotné použití tohoto modelu (rozpoznávání).

Struktura programu je rozdělena do těchto souborů:

- train.py - soubor pro spuštění trénování modelu a vytisknutí výsledků trénování
- nn.py - v tomto souboru je definováno jak vypadá architektura sítě, vytváří se zde samotný model CNN
- load.py - funkce pro načítání dat
- preprocess.py - obsahuje funkce pro předzpracování dat a extrakce příznaků
- save.py - funkce pro ukládání už předzpracovaných dat a modelu
- recognize.py, recognize_plot.py - soubory obsahující skripty pro využití již natrénovaného modelu

1.1 Použité nástroje

Tento systém pro rozpoznávání řeči byl vytvořen v programovacím jazyce Python. Samotné rozpoznávání je založeno na extrakci příznaků pomocí MFCC z důvodu nelineárního vnímání zvuku lidským uchem a pro klasifikaci je použita konvoluční neuronová síť. Pro vytvoření neuronové sítě je použito aplikační rozgraní Keras pro TensorFlow.

Při trénování byla použita volně dostupná datová sada Speech commands, která je navržena tak aby pomáhala navrhovat, trénovat a vyhodnocovat jednoduché systémy pro rozpoznávání řeči. Tato datová sada je vydána pod licencí Creative Commons BY 4.0, toto ji umožňuje snadno začlenit do ukázkových programů a dalšách skriptů, kde je ji možné stáhnout bez zásahu uživatele. Datová sada obsahuje dohromady 105 829 nahrávek 35 slov od různých řečníků v různých prostředích.

Pro trénování navrženého systému jsem vybrala těchto 15 slov: ‘Down’, ‘Five’, ‘Four’, ‘Go’, ‘Left’, ‘No’, ‘Off’, ‘On’, ‘Right’, ‘Seven’, ‘Stop’, ‘Two’, ‘Up’, ‘Yes’, ‘Zero’.

KAPITOLA 2

Programová dokumentace

Tyto balíčky jsou vyžadovány pro spuštění programu:

- numpy, scipy, glob, os, keras, pyaudio, sys, queue, matplotlib, sklearn, random, pandas_ml, argparse, inspect, pyqtgraph

2.1 Načítání dat

Data jsou načtena z adresáře, který se nachází v proměnné path. Struktura tohoto adresáře musí být vždy taková, že obsahuje podadresáře s názvy rozpoznávaných slov a v těchto podadresářích musí být nahrávky jednotlivých slov. Vzhledem k použité datové sadě je potřeba aby délka každé nahrávky byla 1s a vzorkovací frekvence 16KHz.

`load.`**`get_data`**(*path*)
Funkce pro načtení dat a extrakci příanaků

Parametry **`path`** – cesta k datové sadě

Vrací

- a - pole s příznaky jednotlivých slov
- lbl_hot - indexy tříd do kterých patří jednotlivá slova
- i+1 - počet slov
- lbl - seznam rozpoznávaných slov

`load.`**`load_data`**(*path*)
Funkce pro načtení vypočtených příznaků pro jednotlivá slova

Parametry **`path`** – cesta k datové sadě

Vrací

- a - pole s příznaky jednotlivých slov
- lbl_hot - indexy tříd do kterých patří jednotlivá slova
- i+1 - počet slov
- lbl - seznam rozpoznávaných slov

`load.`**`load_dirs`**(*p*)
Funkce pro načtení seznamu adresářů - každý adresář odpovídá jednomu slovu

Parametry **`p`** – cesta k datové sadě

Vrací seznam adresářů seřazený podle abecedy

load.**load_labels**(*path*)
Načtení uloženého seznamu rozpoznávaných slov

Parametry **path** – cesta k datové sadě

Vrací seznam slov

load.**load_wav**(*p*, *w*)
Funkce pro načtení zvukových souboru

Parametry

- **p** – cesta k datové sadě
- **w** – načítané slovo (název adresáře, ve kterém se nachází)

Vrací

- mdata - pole s načtenými zvukovými soubory
- fs - vzorkovací frekvence

2.2 Předzpracování a extrakce příznaků

preprocess.**delta**(*features*)
Funkce pro výpočet dynamických příznaků

Parametry **features** – pole se statickými příznaky

Vrací pole se statickými i dynamickými příznaky

preprocess.**detrend_data**(*data*)
Funkce pro odstranění stejnosměrné složky ze signálu

Parametry **data** – vstupní signál - řeč (slovo)

Vrací signál bez stejnosměrné složky

preprocess.**fft**(*data*, *NFFT*)
Funkce pro výpočet rychlé Fourierovy transformace

Parametry

- **data** – rámce řečového signálu
- **NFFT** – délka rychlé fourierovy transofmace (musí být vždy 2^n)

Vrací výkonové spektrum signálu

preprocess.**filter_banks**(*n_filt*, *pow_f*, *sample_rate*, *NFFT*)
Aplikace banky filtrů na vykonové spektrum signálu

Parametry

- **n_filt** – počet filtrů
- **pow_f** – výkonové spektrum
- **sample_rate** – vzorkovací frekvence
- **NFFT** – délka rychlé fourierovy transofmace (musí být vždy 2^n)

Vrací signál, na který byla aplikována filtrace

preprocess.**hamm**(*fs*, *data*)
Funkce pro aplikování Hammingova okna na vstuní signál

Parametry

- **fs** – vzorkovací frekvence
- **data** – jeden rámec s řečovým sginálem

Vrací signál, na který je aplikováno Hammingovo okno

preprocess.**hamm_all**(*fs*, *data*)
Funkce pro aplikování Hammingova okna na všechny rámce jednoho slova

Parametry

- **fs** – vzorkovací frekvence
- **data** – řečový signál rozdělený na rámce

Vrací pole s rámci, na které bylo aplikováno Hammingovo okno

preprocess.**mfcc**(*fb*)
Funkce pro výpočet disktrétní kosinové transormace nad filtrovaným signálem

Parametry **fb** – signál, na který byla aplikována banka filtrů

Vrací pole se statickými příznaky

preprocess.**pre_emphasis**(*data*)
Funkce pro vyrovnání frekvenční charakteristiky řeči

Parametry **data** – vstupní signál - řeč (slovo)

Vrací signál po průchodu filtrem pro preemfázi

preprocess.**run_preprocess**(*fs*, *data*)
Funkce pro spuštění předzpracování a extrakce příznaků

Parametry

- **fs** – vzorkovací frekvence
- **data** – vstupní řečový signál (slovo)

Vrací pole s příznaky

preprocess.**segmentation**(*step*, *windowsize*, *data*)
Funkce pro rozdělení vstupního signálu na rámce

Parametry

- **step** – posun rámce
- **windowsize** – velikost rámce
- **data** – vstupní signál - řeč (slovo)

Vrací pole s jednotlivými rámci

2.3 Ukládání dat

save.**save_data**(*path*)
Funkce pro výpočet a uložení příznaků pro všechna slova z datové sady

Parametry **path** – cesta k datové satě

save.**save_labels**(*path*)
Funkce pro uložení seznamu slov

Parametry **path** – cesta k datové sadě

2.4 Vytvoření modelu neuronové sítě

nn.**getmodel**(*dense*)
Funkce pro vytvoření a zkompilování modelu neuronové sítě

Parametry **dense** – počet rozpoznáváných tříd (slov)

Vrací vrací model neuronové sítě

2.5 Trénování modelu neuronové sítě

load.**get_data**(*path*)
Funkce pro načtení dat a extrakci příanaků

Parametry **path** – cesta k datové sadě

Vrací

- a - pole s příznaky jednotlivých slov
- lbl_hot - indexy tříd do kterých patří jednotlivá slova
- i+1 - počet slov
- lbl - seznam rozpoznávaných slov

load.**load_data**(*path*)
Funkce pro načtení vypočtených příznaků pro jednotlivá slova

Parametry **path** – cesta k datové sadě

Vrací

- a - pole s příznaky jednotlivých slov
- lbl_hot - indexy tříd do kterých patří jednotlivá slova
- i+1 - počet slov
- lbl - seznam rozpoznávaných slov

load.**load_dirs**(*p*)
Funkce pro načtení seznamu adresářů - každý adresář odpovídá jednomu slovu

Parametry **p** – cesta k datové sadě

Vrací seznam adresářů seřazený podle abecedy

load.**load_labels**(*path*)
Načtení uloženého seznamu rozpoznávaných slov

Parametry **path** – cesta k datové sadě

Vrací seznam slov

load.**load_wav**(*p*, *w*)
Funkce pro načtení zvukových souboru

Parametry

- **p** – cesta k datové sadě

- **w** – načítané slovo (název adresáře, ve kterém se nachází)

Vrací

- mdata - pole s načtenými zvukovými soubory
- fs - vzorkovací frekvence

2.6 Grafická aplikace pro rozpoznávání

class recognize_plot.**Recognizer**(*parent=None*)
Třída pro vytvoření okna s vykreslováním dat z mikrofonu a využití předem natrénovaného modelu pro rozpoznávání

callback(*in_data*, *frame_count*, *time_info*, *status*)
Funkce pro zpracovávání dat z mikrfonu.

Parametry

- **in_data** – vstupní signál z mikrofonu
- **frame_count** – počet rámců
- **time_info** – slovník s klíči pro PortAutdio
- **status** – příznak pro PortAudio

Vrací pokud jsou data prázdná vrací None, jinak příznak, že snímání zvuku může pokračovat

process()
Funkce, která dokud je aktivní snímání dat z mikrofonu beží a vykresluje aktuální průběh.

Zároveň také bere data ze vstupu se vzorkovací frekvencí 16~KHz a po načtení určité části dat provede předzpracování a jeho výsledek předá natrénovanému modelu. Z této predikce je pak vybráno slovo s největší pravděpodobností a to je vypsáno do konzole. Pokud jsou všechny pravděpodobnosti menší než 0.6 tak je slovo klasifikováno jako unknown – nerozpoznáno.

start_stream()
Funkce pro spustění snímání zvuku z mikrofonu

recognize_plot.**run_recognize**()
Funkce pro spuštění celého procesu rozpoznávání.

2.7 Konzolová aplikace pro rozpoznávání

recognize.**callback**(*in_data*, *frame_count*, *time_info*, *status*)
Funkce pro zpracovávání dat z mikrfonu.

Parametry

- **in_data** – vstupní signál z mikrofonu
- **frame_count** – počet rámců
- **time_info** – slovník s klíči pro PortAutdio
- **status** – příznak pro PortAudio

Vrací pokud jsou data prázdná vrací None, jinak příznak, že snímání zvuku může pokračovat

recognize.**run_recognize**()

Funkce, která spouští snímání zvuku z mikrofonu. Poté zde běží smyčka, která dokud je mikrofon aktivní bere data ze vstupu se vzorkovací frekvencí 16~KHz a po načtení určité části dat provede předzpracování, výsledek předá natrénovanému modelu. Z této predikce je pak vybráno slovo s největší pravděpodobností a to je vypsáno do konzole. Pokud jsou všechny pravděpodobnosti menší než 0.6 tak je slovo klasifikováno jako unknown – nerozpoznáno.

KAPITOLA 3

Uživatelská dokumentace

Program je rozdělen do dvou částí, první část je trénování modelu neuronové sítě a druhá složí k automatickému rozpoznávání izolovaných slov.

3.1 Příprava modelu - trénování

Nejdříve je potřeba spustit program train.py s cesou k datové sadě. Jsou tři možné způsoby jak program spustit.

Prvním je spustit program pouze s cestou k datové sadě, která je povinným parametrem, tato možnost načte zvukové soubory, provede předzpracování a extrakci příznaků a natrénuje model.

Druhá možnost je spusit skript s přepínačem -s, kdy se spustí načítání zvukových souborů, předzpracování a extrakce příznaků a tyto příznaky jsou pak uloženy do souborů, poté je také spuštěno trénování modelu. Ukládání příznaků je užitečné při ladění modelu, kdy se nemusí pokaždé provádět zdlouhavá příprava pro trénování, ale stačí pouze načíst připravená data.

Poslední možnost je spustit program s přepínačem -l, který načte příznaky uložené v souborech a spustí trénování modelu.

Nápovědu je možné vypsat spuštěním ‘python3 train.py -h’

3.2 Rozpoznávání

3.2.1 Konzolová aplikace

Tato část programu vyžaduje natrénovaný model a uložený seznam slov v adresáři, kde se nachází právě spouštěný program - to se vytváří automaticky po ukončení trénování modelu. Konzolová aplikace se pak spustí příkazem ‘python3 recognize.py’. Aplikace vyžaduje mikrofon, ze kterého se snímá zvuk. Dále už stačí jen vyslovit slovo a aplikace jej pak vypíše do konzole. Aplikace se ukončí stisknutím klávesové zkratky ‘ctrl + c’.

3.2.2 Grafická aplikace

Grafická aplikace má stejné požadavky jako ta konzolová tzn. vyžaduje natrénovaný model a uložený seznam slov v adresáři, kde se nachází právě spouštěný program. Aplikace se spouští příkazem ‘python3 recognize_plot.py’, po potvrzení tohoto příkazu se spustí okno s vykreslováním aktuální průběhu signálu z mikrofonu a výpisem rozpoznaných slov. Ukončit ji lze zavřením okna, nebo klávesovou zkratkou ‘ctrl + c’ z konzole.

KAPITOLA 4

Indexy a tabulky

- genindex
- modindex

l

load, 6

n

neural_network, 6
nn, 6

p

preprocess, 4

r

recognize, 7
recognize_plot, 7

s

save, 5

C

callback() (metoda recognize_plot.Recognizer), 7
callback() (v modulu recognize), 7

D

delta() (v modulu preprocess), 4
detrend_data() (v modulu preprocess), 4

F

fft() (v modulu preprocess), 4
filter_banks() (v modulu preprocess), 4

G

get_data() (v modulu load), 3, 6
getmodel() (v modulu nn), 6

H

hamm() (v modulu preprocess), 4
hamm_all() (v modulu preprocess), 5

L

load (modul), 3, 6
load_data() (v modulu load), 3, 6
load_dirs() (v modulu load), 3, 6
load_labels() (v modulu load), 4, 6
load_wav() (v modulu load), 4, 6

M

mfcc() (v modulu preprocess), 5

N

neural_network (modul), 6
nn (modul), 6

P

pre_emphasis() (v modulu preprocess), 5
preprocess (modul), 4
process() (metoda recognize_plot.Recognizer), 7

R

recognize (modul), 7
recognize_plot (modul), 7
Recognizer (třída v recognize_plot), 7
run_preprocess() (v modulu preprocess), 5
run_recognize() (v modulu recognize), 7
run_recognize() (v modulu recognize_plot), 7

S

save (modul), 5
save_data() (v modulu save), 5
save_labels() (v modulu save), 5
segmentation() (v modulu preprocess), 5
start_stream() (metoda recognize_plot.Recognizer), 7